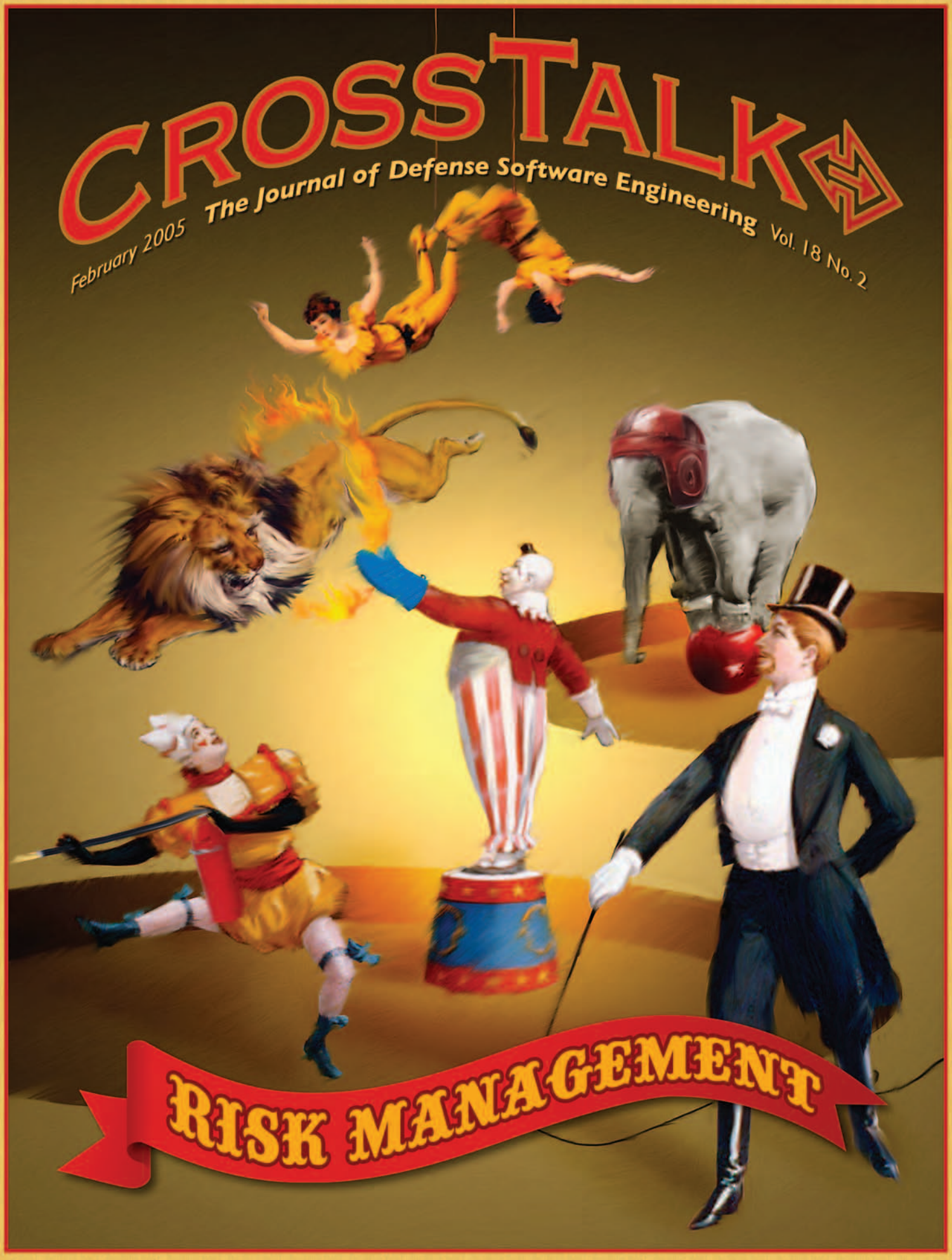

CROSSTALK
The Journal of Defense Software Engineering
February 2005
Vol. 18 No. 2
RISK MANAGEMENT

## Risk Management

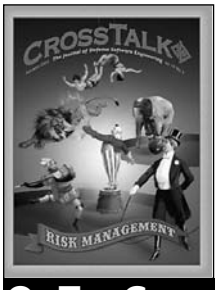


## Best Practices



## Open Forum






**ON THE COVER**
Cover Design by Kent Bingham.

## Departments




# CROSSTALK

| | |
|---|---|
| OC-ALC/ MAS CO-SPONSOR | *Kevin Stamey* |
| OO-ALC/MAS CO-SPONSOR | *Randy Hill* |
| WR-ALC/MAS CO-SPONSOR | *Tom Christian* |
| PUBLISHER | *Tracy Stauder* |
| ASSOCIATE PUBLISHER | *Elizabeth Starrett* |
| MANAGING EDITOR | *Pamela Palmer* |
| ASSOCIATE EDITOR | *Chelene Fortier-Lozancich* |
| ARTICLE COORDINATOR | *Nicole Kentta* |
| CREATIVE SERVICES COORDINATOR | *Janna Kay Jensen* |
| PHONE | (801) 775-5555 |
| FAX | (801) 777-8069 |
| E-MAIL | crosstalk.staff@hill.af.mil |
| CROSSTALK ONLINE | www.stsc.hill.af.mil/ crosstalk |

***Oklahoma City-Air Logistics Center (OC-ALC), Ogden-Air Logistics Center (OO-ALC), and Warner Robins-Air Logistics Center (WR-ALC) MAS Software Divisions*** are the official co-sponsors of CROSSTALK, The Journal of Defense Software Engineering. The MAS Software Divisions and the Software Technology Support Center (STSC) are working jointly to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

***The STSC*** is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal.

S T S C

***Subscriptions***: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 7.

OO ALC/MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

***Article Submissions***: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

***Reprints:*** Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

***Trademarks and Endorsements:*** This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, or the STSC. All product names referenced in this issue are trademarks of their companies.

***Coming Events***: Please submit conferences, seminars, symposiums, etc. that are of interest to our readers at least 90 days before registration. Mail or e-mail announcements to us.

***CrossTalk Online Services:*** See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-7026, or e-mail <stsc.webmaster@hill.af.mil>.

***Back Issues Available:*** Please phone or e-mail us to see if back issues are available free of charge.

# Risk Management Offers Broad Payoffs

Risk management really has no boundaries. Even routine tasks can benefit from maintaining a successful operational risk management (ORM) program. After once missing a flight from Los Angeles due to traffic congestion, I devised a risk management plan for my business travel. I now schedule my flights around rush-hour traffic. On the night before I leave to return home, I transfer to a hotel just outside of the airport. I also ask around to make sure I know how much time it takes to get through ticketing and security at the airport under the new homeland security measures. By thinking through the potential snags in my travel plans, I can avoid a costly delay in reuniting with my family back home.

Over the last several years, the Air Force has required all levels of the agency to implement ORM. Despite the policy and mandatory training, it is still uncommon for organizations to institutionalize risk management, let alone consider simple, day-to-day changes to reduce risk. Many of these organizations stop at a risk management plan for their organization.

Risk management is not a program to fill a policy, a Capability Maturity Model® objective, or other square: It is good business. Mature software organizations serious about managing risk instill processes that manage it at the project level. Key elements included in an effective risk management plan are to implement process at the project level, not just the organizational level; identify who can accept what risk; evaluate probability and consequence; include a mechanism for recurring evaluation of risk; track risk mitigation; and provide ownership to project members for identifying and managing risk.

ORM is a process of identifying and controlling hazards – something each of us deals with daily in our personal lives and at the workplace. As professionals, we owe it to our ultimate customers – the warfighters – to deal effectively with risks and increase the probability of their successful missions.

Kevin Stamey
*Oklahoma City Air Logistics Center, Co-Sponsor*



# Exercising Risk Management Skills

In last month's issue, we published a policy memorandum titled "Revitalizing the Software Aspects of Systems Engineering." Risk management, one of the 10 software focus areas highlighted in this memo, is the theme of this month's issue. Where does one start to revitalize this important management practice? Informing, educating, and reminding your workforce is one place, and CROSSTALK can help. This month we include several articles describing both the basics of an effective risk management process and how a variety of projects are employing and benefiting from risk management. Risk management challenges are also presented, and authors are quick to point out that this practice isn't easy due to the uniqueness of system program risks.

I'd like to share a few thoughts that might be helpful as you evaluate your risk management activities or lack thereof. Take a minute and compare risk management to exercise. You'll probably agree that our personal lives are so busy that we struggle to find even a spare hour to devote to exercise. But we all know it's a continuous requirement for our bodies to be physically fit and healthy. We also hear how exercise is a critical issue with our youth today – they need plenty of exercise, too. So, if you are a project manager and your daily routine seems too full to squeeze in another task, think about setting project time aside to exercise your risk management skills. And don't do it alone; involve your teams and make it a routine for all.

I hope you find this month's issue a good reminder of why practicing risk management is critical to a healthy and successful program.

Tracy Stauder
*Publisher*

# Understanding Risk Management

*Software Technology Support Center*

*The U.S. Air Force's Software Technology Support Center offers an updated and condensed version of the "Guidelines for Successful Acquisition and Management of Software-Intensive Systems" (GSAM) on its Web site <www.stsc.hill.af.mil/resources/tech_docs>. This article is taken from Chapter 5 "Risk Management" of the GSAM (Version 4.0). We are pleased that all editions have been so well received and that many individuals and programs have worked hard to implement the principles contained therein. The latest edition provides a usable desk reference that gives a brief but effective overview of important software acquisition and development topics, provides checklists for rapid self-inspection, and provides pointers to additional information on the topics covered.*

Risk is a product of the uncertainty of future events and is a part of all activity. It is a fact of life. We tend to stay away from situations that involve high risk to things we hold dear. When we cannot avoid risk, we look for ways to reduce it or its impact upon our lives. Yet even with careful planning and preparation, risks cannot be completely eliminated because they cannot all be identified beforehand. Even so, risk is essential to progress.

The opportunity to succeed also carries the opportunity to fail. It is necessary to learn to balance the possible negative consequences of risk with the potential benefits of its associated opportunity [1]. Risk may be defined as the possibility to suffer damage or loss. The possibility is characterized by three factors [1]:

1. The probability or likelihood that loss or damage will occur.
2. The expected time of occurrence.
3. The magnitude of the negative impact that can result from its occurrence.

The seriousness of a risk can be determined by multiplying the probability of the event actually occurring by the potential negative impact to the cost, schedule, or performance of the project:

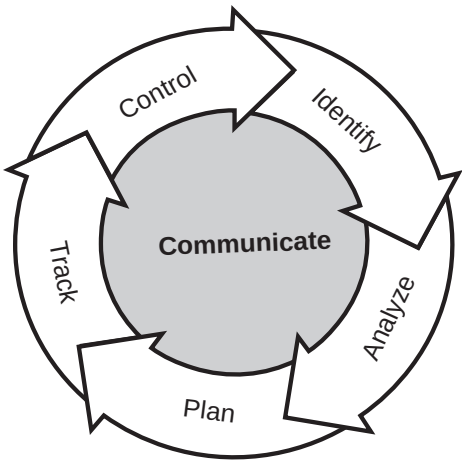
**Risk Severity = Probability of Occurrence x Potential Negative Impact**

Thus, risks where probability of occurrence is high and potential impact is very low, or vice versa, are not considered as serious as risks where both probability of occurrence and potential impact are medium to high.

Project managers recognize and accept the fact that risk is inherent in any project. They also recognize that there are two ways of dealing with risk. One, risk management, is proactive and carefully analyzes future project events and past projects to identify potential risks. Once risks are identified, they are dealt with by taking measures to reduce their probability or to reduce their impact. The alternative to risk management is crisis management. It is a reactive and resource-intensive process, with available options constrained or restricted by events [1].

Effective risk management requires establishing and following a rigorous process. It involves the entire project team, as well as requiring help from outside experts in critical risk areas (e.g., technology, manufacturing, logistics, etc.). Because risks will be found in all areas of the project and will often be interrelated, risk management should include hardware, software, integration issues, and the human element [2].

Figure 1: *Software Engineering Institute's Risk Management Paradigm [3]*




## Process Description

Various paradigms are used by different organizations to coordinate their risk management activities. A commonly used approach is shown in Figure 1. While there are variations in the different paradigms, certain characteristics are universally required for the program to be successful [2]:

- The risk management process is planned and structured.
- The risk process is integrated with the acquisition process.
- Developers, users, procurers, and all other stakeholders work together closely to implement the risk process.
- Risk management is an ongoing process with continual monitoring and reassessment.
- A set of success criteria is defined for all cost, schedule, and performance elements of the project.
- Metrics are defined and used to monitor effectiveness of risk management strategies.
- An effective test and evaluation program is planned and followed.
- All aspects of the risk management program are formally documented.
- Communication and feedback are an integral part of all risk management activities.

While your risk management approach should be tailored to your project needs, it should incorporate these fundamental characteristics. The process is iterative and should have all the components shown in Figure 2. Note that while planning appears as the first step, there is a feedback loop from the monitoring activity that allows planning and the other activities to be redone or controlled by actual results, providing continual updates to the risk management strategy. In essence, the process is a standard approach to problem solving:

1. Plan or define the problem-solving process.
2. Define the problem.
3. Work out solutions for those problems.
4. Track the progress and success of the solutions.

The following sections expand upon the risk management approach.

### Planning

Risk planning includes developing and documenting a structured, proactive, and comprehensive strategy to deal with risk. Key to this activity is establishing methods and procedures to do the following:

1. Establish an organization to take part in the risk management process.
2. Identify and analyze risks.
3. Develop risk-handling plans.
4. Monitor or track risk areas.
5. Assign resources to deal with risks.

A generic sample risk management plan can be found in Appendix B of the "Risk Management Guide for DoD Acquisition" [4].

### Assessment

Risk assessment involves two primary activities: risk identification and risk analysis. Risk identification is actually begun early in the planning phase and continues throughout the life of the project. The following methods are often used to identify possible risks [1]:

- Brainstorming.
- Evaluations or inputs from project stakeholders.
- Periodic reviews of project data.
- Questionnaires based on taxonomy, the classification of product areas and disciplines.
- Interviews based on taxonomy.
- Analysis of the Work Breakdown Structure.
- Analysis of historical data.

When identifying a risk it is essential to do so in a clear and concise statement. It should include three components [1]:

1. **Condition:** A sentence or phrase briefly describing the situation or circumstance that may have caused concern, anxiety, or uncertainty.
2. **Consequence:** A sentence describing the key negative outcomes that may result from the condition.
3. **Context:** Additional information about the risk to ensure others can understand its nature, especially after the passage of time.

Table 1 is an example of a risk statement [1].

The other half of assessment is risk analysis. This is the process of examining each risk to refine the risk description, isolate the cause, quantify the probability of occurrence, and determine the nature and impact of possible effects. The result of this process is a list of risks rated and prioritized according to their probability of occurrence, severity of impact, and relationship to other risk areas [2].

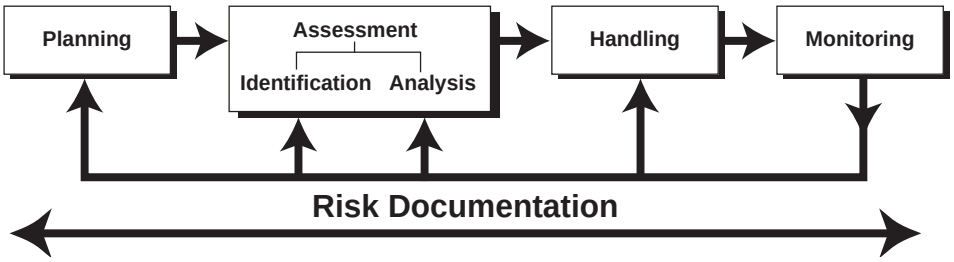



Figure 2: *Risk Management Process Example*

| | |
|---|---|
| **Condition** | End users submit requirements changes even though we are in the design phase and the requirements have been baselined. |
| **Consequence** | Changes could extend system design cycle and reduce available coding time. |
| **Probability and Impact** | 80%. $2 million. |
| **Mitigation Actions** | Who, what, and when? |

Table 1: *Risk Statement Example*

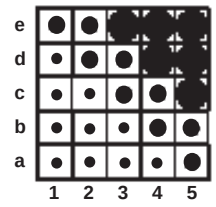
Once risks have been defined, and probability of occurrence and consequences assigned, the risk can be rated as to its severity. This facilitates prioritizing risks and deciding what level of resources to devote to each risk. Figure 3 depicts an assessment model using risk probability and consequence levels in a matrix to determine a level of risk severity. In addition to an overall method of risk rating, the model also gives good examples of probability levels and types and levels of consequences. The ratings given in the assessment guide matrix are suggested minimum ratings. It may be necessary to adjust the moderate and high thresholds to better coincide with the type of project.

Figure 3: *Defense Acquisition University Assessment Model [4]*

**PROBABILITY**

| Level | Likelihood Risk Will Happen |
|---|---|
| a | Minimal/Remote |
| b | Small/Unlikely |
| c | Probable/Likely |
| d | Large/Highly Likely |
| e | Significant/Near Certainty |

**ASSESSMENT GUIDE**




**RISK IMPACT RATING**

| | |
|---|---|
| ● | **HIGH:** Significant impact on cost, schedule, and performance. Significant action required. High priority management attention required. |
| ● | **MODERATE:** Some impact. Special action may be required. Additional management attention may be needed. |
| • | **LOW:** Minimum impact. Normal oversight needed to ensure risk remains low. |

**CONSEQUENCE**

| Level | Technical Performance | Schedule | Cost | Impact on Other Teams |
|---|---|---|---|---|
| 1 | Minimal or No Impact | Minimal or No Impact | Minimal or None | None |
| 2 | Acceptable With Some Reduction in Margin | Additional Resources Required – Able to Meet Need Dates | < 5% | Some |
| 3 | Acceptable With Significant Reduction in Margin | Minor Slip in Key Milestone – Not Able to meet Need Dates | 5% - 7% | Moderate |
| 4 | Acceptable – No Remaining Margin | Minor Slip in Key Milestone or Critical Path Impacted | 7% - 10% | Major |
| 5 | Unacceptable | Cannot Achieve Key Team of Major Project Milestone | > 10% | Unacceptable |


Figure 4: *Risk Handling Process*

### *Handling*

Risk handling is the process that identifies, evaluates, selects, and implements options for mitigating risks, as shown in Figure 4. Two approaches are used in handling risk. The first is to employ options that reduce the risk itself. This usually involves a change in current conditions to lessen the probability of occurrence. The second approach, often employed where risk probability is high, is to use options that reduce the negative impact to the project if the risk condition should occur. Improving jet engine maintenance and inspection procedures to reduce the risk of in-flight engine failure is an example of the first approach. Providing a parachute for the pilot, to reduce loss if the risk condition should occur, is an example of the second approach.

### *Monitoring*

Risk monitoring is the process of continually tracking risks and the effectiveness of risk handling options to ensure risk conditions do not get out of control. This is done by knowing the baseline risk management plans, understanding the risks and risk handling options, establishing meaningful metrics, and evaluating project performance against the established metrics, plans, and expected results throughout the acquisition process. Continual monitoring also enables new risks to be identified if they become apparent over time. Monitoring further reveals the interrelationships between various risks [2].

The monitoring process provides feedback into all other activities to improve the ongoing, iterative risk management process for the current and future projects.

### *Documentation*

Risk documentation is absolutely essential for the current, as well as future, projects. It consists of recording, maintaining, and reporting risk management plans, assessments, and handling information. It also includes recording the results of risk management activities, providing a knowledge base for better risk management in later stages of the project and in other projects [2]. Documentation should include – as a minimum – the following information:

- Risk management plans.
- Project metrics to be used for risk management.
- Identified risks and their descriptions.
- The probability, severity of impact, and prioritization of all known risks.
- Description of risk handling options selected for implementation.
- Project performance assessment results, including deviations from the baseline plans.
- Records of all changes to the above documentation, including newly identified risks, plan changes, etc.

## Risk Management Checklist

This checklist is provided to assist you in risk management. If you answer no to any of these questions, you should examine the situation carefully for the possibility of greater risks to the project. This is only a cursory checklist for such an important subject. Please see [5, 6] for more detailed checklists.

- ❑ Do you have a comprehensive, planned, and documented approach to risk management?
- ❑ Are all major areas/disciplines represented on your risk management team?
- ❑ Is the project manager experienced with similar projects?
- ❑ Do the stakeholders support disciplined development methods that incorporate adequate planning, requirements analysis, design, and testing?
- ❑ Is the project manager dedicated to this project, and not dividing his or her time among other efforts?
- ❑ Are you implementing a proven development methodology?
- ❑ Are requirements well defined, understandable, and stable?
- ❑ Do you have an effective requirements change process in place, and do you use it?
- ❑ Does your project plan call for tracking/tracing requirements through all phases of the project?
- ❑ Are you implementing proven technology?
- ❑ Are suppliers stable, and do you have multiple sources for hardware and equipment?
- ❑ Are all procurement items needed for your development effort short lead-time items (no long-lead items)?
- ❑ Are all external and internal interfaces for the system well defined?
- ❑ Are all project positions appropriately staffed with qualified, motivated personnel?
- ❑ Are the developers trained and experienced in their respective development disciplines (i.e., systems engineering, software engineering, language, platform, tools, etc.)?
- ❑ Are developers experienced or familiar with the technology and the development environment?
- ❑ Are key personnel stable and likely to remain in their positions throughout the project?
- ❑ Is project funding stable and secure?
- ❑ Are all costs associated with the project known?
- ❑ Are development tools and equipment used for the project state-of-the-art, dependable, and available in sufficient quantity, and are the developers familiar with the development tools?
- ❑ Are the schedule estimates free of unknowns?
- ❑ Is the schedule realistic to support an acceptable level of risk?
- ❑ Is the project free of special environmental constraints or requirements?
- ❑ Is your testing approach feasible and appropriate for the components and system?
- ❑ Have acceptance criteria been established for all requirements and agreed to by all stakeholders?
- ❑ Will there be sufficient equipment to do adequate integration and testing?
- ❑ Has sufficient time been scheduled for system integration and testing?
- ❑ Can software be tested without complex testing or special test equipment?
- ❑ Is a single group in one location developing the system?
- ❑ Are subcontractors reliable and proven?
- ❑ Is all project work being done by groups over which you have control?
- ❑ Are development and support teams all collocated at one site?
- ❑ Is the project team accustomed to working on an effort of this size (neither bigger nor smaller)?

## Summary

Project managers recognize and accept the fact that risk is inherent in any project. The most successful project managers choose to deal proactively with risk. They carefully analyze future project events and past projects to identify potential risks. Once risks are identified, managers take steps to reduce their probability or reduce the impact associated with them by establishing and following a

rigorous process, which involves the entire project team as well as outside experts. Risk management should include hardware, software, integration issues, and the human element. A risk management process includes planning, assessment, handling, monitoring, and documentation. Risk is a product of the uncertainty of future events and is a part of all activity. Learning to balance its possible negative consequences with its potential benefits is the key to successful risk management.◆

## References


1. Software Technology Support Center. "Life Cycle Software Project Management." Project Initiation. Hill Air Force Base, UT, 9 Oct. 2001.
2. Department of Defense. "Risk Management Guide for DoD Acquisition." Washington, D.C.: DoD Feb. 2001: Chap. 2 <www.dsmc.dsm.mil/pubs/gdbks/risk_management.htm>.
3. Higuera, Ron, and Yacov Haimes. "Software Risk Management." Pittsburgh, PA: Software Engineering Institute, 28 June 1996 <www.sei.cmu.edu/publications/documents/96.reports/96.tr.012.html>.
4. Department of Defense. "Risk Management Guide for DoD Acquisition." Washington, D.C.: DoD, Feb. 2001: Appendix B <www.dsmc.dsm.mil/pubs/gdbks/risk_management.htm>.
5. Arizona State University. "Question List for Software Risk Identification in the Classroom." <www.eas.asu.edu/~riskmgmt/qlist.html>.
6. Department of Energy. Risk Assessment Questionnaire. <http://cio.doe.gov/sqse/pm_risk.htm>.


## About the Author


The **Software Technology Support Center** (STSC) produced the "Guidelines for Successful Acquisition and Management of Software-Intensive Systems." Visit the STSC Web site at <www.stsc.hill.af.mil/resources/tech_docs> to access all 17 chapters of this document. The STSC is dedicated to helping the Air Force and other U.S. government organizations improve their capability to buy and build software better. The STSC provides hands-on assistance in adopting effective technologies for software-intensive systems. The STSC helps organizations identify, evaluate, and adopt technologies that improve software product quality, production efficiency, and predictability. Technology is used in its broadest sense to include processes, methods, techniques, and tools that enhance human capability. The STSC offers consulting services for software process improvement, software technology adoption, and software technology evaluation, including the Capability Maturity Model® Integration, software acquisition, project management, risk management, cost and schedule estimation, configuration management, software measurement, and more.

**Software Technology Support Center**
**6022 Fir AVE BLDG 1238**
**Hill AFB, UT 84056-5820**
**Phone: (801) 586-0154**
**DSN: 586-0154**
**E-mail: stsc.consulting@hill.af.mil**

# Risk Management for Systems of Systems©

Dr. Edmund H. Conrow
*Risk-Services.Com*

*Systems of systems (SOS) present some unusual risk management challenges that often are not explicitly addressed, yet can impact the resulting degree of system effectiveness. Potential risks associated with integrating a diverse set of systems and associated hardware/hardware, hardware/software, and software/software often exist; these are made all the more difficult by individual systems at different levels of maturity and potential risks that do not exist at the individual system level. Established risk management processes may be in place for different systems, yet process steps and associated tools and techniques may not be compatible. This article briefly examines some key SOS risk management process issues together with methods for better implementing risk management on such programs.*

Most of the literature and government guidance to date on project risk management has been focused on individual programs or systems. Yet systems of systems are becoming more complex and more commonplace in the United States and abroad.

One system of systems that many people in the United States have used without recognizing it is the air traffic control system. In the United States, according to the General Accounting Office (GAO):

> … the en route centers (of the Air Traffic Control system) alone rely on over 50 systems to perform mission-critical information processing and display, navigation, surveillance, communications, and weather functions. [1]

A current Department of Defense (DoD) example of a complex system of systems under development is the Future Combat System (FCS). In building FCS, the GAO says:

> … Army leaders decided to include interoperability with other systems in the FCS design, and design the individual FCS systems to work as part of a networked system of systems with a first-of-a-kind network. [2]

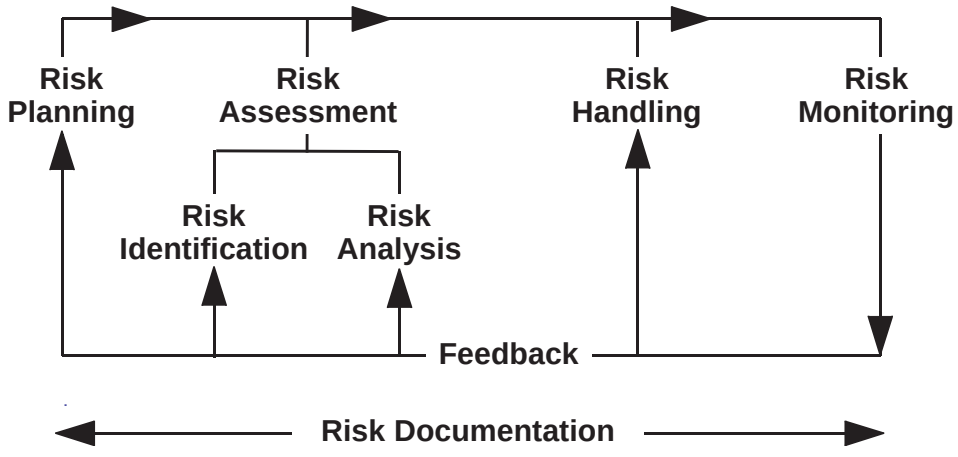
For FCS, the GAO writes:

> … 14 major weapon systems or platforms have to be designed and integrated simultaneously and within strict size and weight limitations in less time than is typically taken to develop, demonstrate, and field a single system. At least 53 technologies that are considered critical to achieving critical performance capabilities will need to be matured and integrated into the system of systems. And the development, demonstration, and production of as many as 157 complementary systems will need to be synchronized with FCS content and schedule. [3]

In this article, I will provide an overview of the risk management process and explore risks that are common to many systems of systems (SOS) implementations along with recommendations for addressing each risk.



Figure 1: *Risk Management Process*




## Risk Management Introduction

*Risk management* is the act or practice of dealing with risk. It includes planning for risk, assessing (identifying and analyzing) risk issues, developing risk handling options, monitoring risks to determine how they have changed, and documenting the overall risk management program. A simplified risk management process flow is given in Figure 1 [4, 5].

- **Risk planning** is the process of developing and documenting an organized, comprehensive, and interactive strategy and methods for identifying risk issues, performing risk analyses, developing and implementing risk handling plans, and monitoring the performance of risk handling actions.
- **Risk assessment** is the process of identifying and analyzing program areas and critical technical process risks to increase the likelihood of meeting cost, performance, and schedule objectives. *Risk identification* is the process of examining the program areas and each critical technical process to identify and document the associated risk. *Risk analysis* is the process of examining each identified risk issue or process to refine the description of the risk, isolating the cause and determining the effects.
- **Risk handling** is the process that identifies, evaluates, selects, and implements options in order to set risk at acceptable levels given program constraints and objectives. This includes the specifics on what should be done, when it should be accomplished, who is responsible, and what are the associated cost and schedule. Risk handling options include assumption, avoidance, control (also

known as mitigation), and transfer. The most desirable handling option is selected and a specific implementation approach is then developed for this option and documented in a risk-handling plan.

- **Risk monitoring** is the process that systematically tracks and evaluates the performance of risk handling actions against established metrics throughout the acquisition process, and provides inputs to update risk-handling strategies as appropriate. Risk monitoring also provides risk-related information to the other processing steps via the feedback function (as illustrated in Figure 1).
- **Risk documentation** is recording, maintaining, and reporting assessments; handling analysis and plans; and monitoring results. It includes all plans, reports for the program manager and decision authorities, and reporting forms that may be internal to the program.

While the above items (with the exception of risk documentation) are related to specific process steps, it is equally important that risk management is properly implemented following appropriate human and organizational behavioral considerations. For example, both *top-down* (program manager lead) and *bottom-up* (worker-level daily performance) are necessary to provide a suitable environment for effective risk management. It is all too common that upper management is disinterested in risk management or sends mixed messages to working-level personnel. Yet, without working-level personnel assimilating risk management principles into their daily job function, it will be difficult at best to have successful risk management.

In general, it is more important and more difficult to create the proper culture on a program to inculcate risk management than it is to master the tools and techniques for the process steps.

## System-of-System Issues

I will now briefly discuss seven relatively common issues for SOS risk management [6], which are given in Table 1. (See Boehm, et. al. [7] for a discussion of some software-intensive SOS risks.) The format used in each case first describes and frames the issue and is then followed by recommended approaches for addressing each issue.

### *1. Multiple Stakeholders*

Multiple buyers, sellers, and other stakeholders will generally exist, and the behavior of each group is not homogeneous. The objective function associated with cost, performance, and schedule (CPS) will be different for different parties. These differences will often lead to contention and potentially sub-optimal design solutions, funding allocation, schedule priority, and increased risk [8].

| Number | Issue | Issue Summary |
|---|---|---|
| 1 | Multiple Stakeholders | Differences in stakeholder's behaviors will often lead to contention and potentially sub-optimal design solutions, funding allocation, schedule priority, and increased risk. |
| 2 | Multiple Risk Management Processes | Differences in risk management processes and their implementation can lead to the omission of risks as well as exaggeration of other risks. |
| 3 | Long Life Cycles | Non-uniform acquisition maturity potentially complicates risk management. |
| 4 | Common Technical Risk Classes | Technical risks are often examined, evaluated, and managed separately, which may not provide insight into potential strengths/surpluses and weaknesses/shortfalls. |
| 5 | Integration Risk | Integration risk is often not explicitly evaluated. |
| 6 | Functional Performance Risk | Functional performance risk is often not explicitly evaluated. |
| 7 | Interface Complexity | It is generally difficult to evaluate interface complexity and accurately relate it to risk. |

Table 1: *Common Systems of Systems Risk Management Issues*

For SOS, multiple prime contractors may exist at the individual system level; these contractors are both buyers from lower-level contractors on an individual program and sellers to both the systems of systems lead contractor and the government. Hence, a variety of objective functions will typically exist at the systems of systems level and reflect different preferences for CPS and associated risk.

In the development of government systems, both the buyer (e.g., government) and seller (e.g., contractor) typically favor increased levels of performance, while the buyer often favors decreased cost and schedule, and the seller favors increased cost and schedule. A common result of this imbalance in both DoD and NASA programs is that performance is the dominant variable and cost and/or schedule are adjusted during the course of the development phase to meet performance requirements [8]. Issues resulting from sub-optimal CPS trades often translate to considerable risk when they are discovered late in the development phase because there is limited ability to efficiently modify designs, etc.

One method to alleviate such problems is to systematically investigate CPS and associated risk in all CPS trades, not just one or two of the three dimensions. Furthermore, the three dimensions of risk should be integrated along with CPS trades to yield a cohesive representation of the potential solution space. In systems of systems, it is common to find marginal risk management focused on technical risk, and weak cost- and schedule-related risk management.

An aid to effective risk management is to have suitable CPS risk management implementation and integration through a central risk management process for each program, as well as at the SOS level. (It is surprisingly common to find separate pockets of CPS risk management within a large-scale program, often with limited program-level integration. This behavior is counter-productive and can lead to weak risk management.)

In addition, differences in the party's objective functions and resulting behaviors should be recognized to avoid *surprises*, balance risk across systems, and to help facilitate mutual awareness and the development of potential solutions prior to risk issues becoming problems later in the program. (Note: a problem is defined here as a risk issue that has occurred [probability = 1].)

For example, a system under development by one government organization often reported risk levels for challenging subsystems that were lower than similar subsystems under development by a different organization. (Here, both systems were in competition with each other and only one would potentially be deployed.) After some time, the government organization responsible for SOS integration instructed the two other government organizations that credible risk analysis results and risk handling plans were far more important than artificially low risk scores.

This *message* was received by both government developmental organizations, and helped to level the playing field between them. This led to increased risk management effectiveness at the SOS organization because fewer resources were needed to evaluate and correct the imbalance in risk analysis results.

### *2. Multiple Risk Management Processes*

Multiple risk management processes will generally exist for SOS. These processes should be, but are often not highly compatible. Without particular attention, this can contribute to weak or ineffective risk management. Risk management differences between systems – and possibly organizations associated with a given system – will likely occur in risk identification and analysis methodology, and the development of risk-handling plans. These process and associated implementation differences can lead to omission of some risks as well as the exaggeration of other risks.

I will now briefly address some common issues associated with different risk management process steps when multiple risk management processes exist.

#### Unstructured and Incomplete Risk Identification

Risk identification is often performed in an unstructured manner and typically uses a small subset of available approaches. The result of these shortcomings is that potential risk issues can be missed and may become problems later in the program.

At least six different risk identification approaches exist such as those based upon the work breakdown structure (WBS), requirements flow-down, and key process evaluation; each of these approaches should be considered [5]. Typically only a few of these methods are used on a large-scale program, yet each should be considered. This shortfall may in part be related to an organization's risk management heritage. That is, organizations with a strong focus on process-level risks (e.g., design and test) may have limited experience with the WBS approach.

In addition, for SOS, a methodology should be used to perform a top-level risk evaluation for each individual program as well as across the programs for items not associated with lower WBS levels. For example, a program is top-level (WBS 1) for its system. However, a particular program is likely second- or third-level WBS for SOS (whose top-level is WBS 1). Some candidate risks may exist at higher WBS levels (e.g., 1-3) and may not manifest in an easily recognizable manner or even exist at lower WBS levels.

Other potential risks may be better addressed across programs at a top level within the SOS and not at lower levels within the individual systems. For example, networking architectures should be addressed at the SOS level (top-down).

#### Differences in Risk Analysis Methodologies

A variety of risk analysis methodologies will typically exist for SOS. This can be problematic since variations in the resulting risk levels for the same item evaluated by different organizations or across different programs may be non-trivial (e.g., vary by one or more risk levels). When different organizations evaluate the same risk issue, a significant difference in estimated risk level may result due to differences in how they perceive risk (e.g., risk tolerance) as well as from using different methodologies.

The organization(s) responsible at the SOS level may have to develop a *Rosetta stone* to compare risk analysis results between organizations and translate results at a lower WBS level to a higher WBS level. Likewise, such organizations should evaluate key risks at the individual program level as well as across programs when possible to ensure that appropriate and consistent risk levels exist.

*"In general, it is more important and more difficult to create the proper culture on a program to inculcate risk management than it is to master the tools and techniques for the process steps."*

#### Unfocused Risk Handling Strategies

Risk handling strategies are often developed in an ad hoc manner and without regard to strategies in place for other risks. A focused risk handling strategy should be used for each risk that management (e.g., risk management board) chooses to address. The strategy should evaluate possible options (assumption, avoidance, control, transfer), select the best option, and then develop the most appropriate implementation approach for that option.

This approach should be used at both the individual program and SOS level. In addition, a *top-level* examination of risk handling strategies across programs should be performed to identify resources that may be applied from one strategy to another, as well as potential constraints across strategies on the quantity and timing of resources available. (See the related discussion in the "Common Technical Risk Classes" section.)

### *3. Long Life Cycles*

SOS can be expected to have long life cycles, ranging from many years to decades. The individual programs may have different levels of maturity varying from early development to operations/support. The resulting non-uniform acquisition maturity potentially complicates risk management at the SOS level. For example, the resulting interactions and integration of some programs in early to mid development and others fielded (thus in operations and maintenance) are often with risk.

Conversely, fielded systems often pose constraints on developmental systems from a SOS perspective because of the integration and operations framework that is developed. However, developmental systems may impact fielded systems within a system of systems due to unanticipated programmatic and/or technical issues that may result.

The risk management process should be tailored to each program within the systems of systems and each corresponding program phase. In addition, the risk management process at the SOS level should not be static but should evolve over time as individual program maturity and the overall level of integration increases, as new systems are added and as additional data is available.

Risk issues that exist and the level of information available about specific risks will vary from early development to operations/support. For example, non-trivial, architecture-level design and technology problems may manifest in early to mid development, while manufacturing and integration problems may be present in mid to later development, and support-related problems may follow system deployment.

Each of the resulting risk issues should be evaluated in the early development phase as part of the trade process and in later program phases as appropriate in order to address them before they become problems. The risk handling plan content and implementation schedule will vary with acquisition, resource availability, and time-urgency considerations during the course of the acquisition cycle. In addition, relatively little information may exist for some risk issues early in the development phase, and the result-

ing uncertainty in the estimated risk level may be non-trivial. The quality of information available and the level of certainty should increase during the course of the program and lead to improved risk handling actions (all else held constant).

### *4. Common Technical Risk Classes*

While technical risks are often examined, evaluated, and managed separately, a finite number of technical risk classes often exist in a given program. Grouping technical risks into risk classes can provide program decision-makers with insight into potential strengths/surpluses and weaknesses/shortfalls associated with processes, personnel, other resources, etc.

Some common technical risk classes often include but are not limited to design, functional performance, integration, resource availability, support, and technology. Broadly speaking, many types of risk outside of pure programmatic entities (e.g., cost and schedule) may be classified as technical risk. Technical risk classes can exist from low WBS levels to the program level (WBS level 1) or SOS level. It is common that several of these risk classes are not explicitly evaluated during the course of the program. I will now briefly discuss how common technical risk classes can be addressed in different risk management process steps.

#### Risk Planning

At the individual program level as well as the SOS level, potential risk classes should be explicitly identified as part of the risk planning process, included in the Risk Management Plan (or equivalent), and updated as warranted. This is important since the common practice of selecting risk classes during risk identification oftentimes leads to some risk classes and corresponding candidate risks being omitted.

#### Risk Identification

A risk identification framework should be used that incorporates standard techniques (e.g., WBS level, requirements flow-down, and key processes) that are selected and adjusted by risk class and program phase. For example, an initial review of key processes (e.g., design, manufacturing, and test) should be performed early in the development phase to identify potential risks. This review should be updated and expanded during the development phase to provide sufficient opportunity to address shortfalls and increase maturity prior to critical program need.

Technology risk, however, is better addressed at the WBS level. This evaluation should be initiated early in the development phase and continued during the development phase until the technology has matured to a satisfactory degree.

#### Risk Analysis

Tailored risk analysis methodologies should be available for specific risk classes. For example, it is generally not sufficient to use a single, generic, probability of occurrence scale (e.g., very high = $E$ to very low = $A$ where $E > A$) when performing a technical risk analysis because many risk issues (e.g., development maturity) cannot be readily framed into a question associated with probability level.

*"... the risk management process at the SOS level should not be static but should evolve over time as individual program maturity and the overall level of integration increases, as new systems are added and as additional data is available."*

For example, if a hardware unit in the early developmental stage exists and a fully operational unit is desired using a generic probability of occurrence scale (as above), this can lead to substantial uncertainty as to what level should be selected, and potentially erroneous results. In this particular example, ordinal probability of occurrence scales tailored to unit maturity (e.g., scientific research = $E$ to fully operational = $A$) and other potential risk classes (e.g., manufacturing) are often much better suited and can help reduce the level of misscoring and provide more consistent results.

(Note: maturity-based scales, such as Technology Readiness Levels [TRL], *do not* estimate risk, but only one component of the probability of occurrence term. Risk is the product or combination of probability of occurrence and consequence of occurrence. Since TRL and other such scales are unrelated to consequence of occurrence, they do not in and of themselves provide an estimate of risk.)

#### Risk Handling

Risk handling strategies should be overlaid for common risk classes across WBS levels at the individual program level and the SOS level to identify potential resource issues in a timely manner. For example, if high-performance custom microelectronic components are needed there may be a limited number of suppliers capable of developing and fabricating such parts. If individual orders are examined within a program, the resulting number of different devices may be small, but when examined across programs the quantity may lead to supplier resource shortfalls (e.g., workstations, software licenses, trained personnel, and fabrication, test, and screening capacity) and contention for these resources.

At the individual program level, there may be no apparent risk, but when viewed at the SOS level the resource-related risk may be considerable. This is all the more important if the supplier has fundamental process difficulties in design, testing, or manufacturing because an issue affecting parts for one program may also impact the SOS level or in some cases an entire industry. In such cases, it may be necessary to understand common resources at the supplier level and prioritize potential needs across the program, SOS, or even industry to reduce the level of potential risk whenever possible.

### *5. Integration Risk*

Integration risk is present on many types of programs and is pervasive on SOS by its very nature, yet is often not explicitly evaluated. Hardware/hardware, hardware/software, and software/software are common forms of integration risk. Multiple layers of integration risk are also common, from low to high WBS levels (e.g., 5 to 1) but also across programs for systems of systems. In addition, new forms of integration risk such as net-based integration issues not commonly seen at the individual program level may occur at the SOS level.

The potential level of integration risk is often substantial because of a tendency to underestimate integration difficulty, and simultaneously overestimate the maturity of items that require integration. This is all the more problematic when integration risks manifest late in a program because the ability to trade CPS is typically limited versus manifesting earlier in the program. The result for govern-

ment programs (e.g., DoD and NASA) is often non-trivial cost and/or schedule growth, while performance degradation are typically small [8].

One helpful strategy for alleviating integration risk is to increase attention to potential integration issues throughout the life cycle – beginning in early development rather than focusing on them late in the development process. This can include using adaptable acquisition models (e.g., spiral), carefully developed interface control documents, and early prototyping and perceptive testing to identify potential issues early when there is greater flexibility to trade CPS.

In addition, the transfer risk handling option should be considered for integration issues – do not simply default to the control (mitigation) option. Oftentimes, the transfer option is thought to be limited to insurance, guarantees, warranties, and similar approaches when it also encompasses a variety of other methods such as transferring risk between interfaces, hardware and software, different organizations (e.g., prime versus subcontractor), and even programs. In some cases, this option may alleviate the level of potential risk (e.g., an inexperienced contractor passing real-time software development to a teammate with considerable experience in this area), so long as the recipient actively works the potential risk rather than passively accepting it.

### *6. Functional Performance Risk*

SOS level functional performance risk may include the ability to demonstrate that desired functions or requirements can be met to a specified performance level. This is a different and somewhat converse concept than design risk, which generally assumes that a requirement *can be met* by the nature of the design. Functional performance risk is rarely estimated, yet functional performance shortfalls can translate to problems late in the program if insufficient progress has been made in demonstrating the performance level of key functions that can be achieved.

The probability of occurrence term of functional performance is often maturity based – and scales that incorporate, for example, unverified analytic modeling to in-field testing from less to more mature might represent a coarse ordinal sequence for use. Initial modeling, simulation, and emulation followed by appropriate incremental demonstrations, prototyping, and testing can be helpful throughout the development and integration cycle to potentially reduce functional performance risk to an acceptable level. Whenever possible, avoid an *all or nothing* demonstration and testing approach late in the program since this will often fall short of achieving necessary performance levels and permit little time for recovery versus an incremental approach maintained during the development phase.

### *7. Interface Complexity*

Complex hardware and software interfaces will often exist within individual programs as well as in SOS. While there may be a desire to explicitly treat complexity in a risk analysis, it is generally difficult to accurately relate complexity to risk. Furthermore, efforts to estimate the risk of interface complexity directly may lead to uncertain, subjective, and/or erroneous results.

Interface complexity is typically related to the probability of occurrence term of risk and unrelated to consequence of occurrence. However, it is generally very difficult to develop specific relationships between complexity and probability of occurrence. While the notion that more complex interfaces should have a higher probability of occurrence (all else held constant) is often reasonable from a qualitative or ordinal sense, it may not be possible to confidently say how much higher the resulting probability level is than an interface with a lower complexity level, and inaccurate and/or uncertain estimates may result. Instead, the analyst should consider whether or not interface complexity could be mapped to other technical risk classes that can then be more readily evaluated. These risk classes can include, but are not limited to, design, integration, and support risk. (See the discussion associated with integration risk.)

## Conclusion

Complex technical and implementation issues will exist for SOS that may be far more difficult to deal with than for simpler implementations or individual programs. Risk management can play a key role in addressing many such issues. The seven risk management issues and recommendations for addressing them presented here are applicable to a variety of SOS and provide a starting point to the reader to apply to their programs.◆

## References


1. United States General Accounting Office. Air Traffic Control. GAO/AIMD-97-30. Washington, DC: GAO, Feb. 1997: 20.
2. United States General Accounting Office. FCS Program Issues. GAO-03-1010R. Washington, DC: GAO, 13 Aug. 2003: 2.
3. United States General Accounting Office. The Army's Future Combat Systems' Features, Risks, and Alternatives. GAO-04-635T. Washington, D.C.: GAO, 1 Apr. 2004: 9.
4. Department of Defense. Risk Management Guide to DoD Acquisition. 5th ed. Vers. 2.0 Ft. Belvoir, VA: Defense Acquisition University, June 2003 <www.dau.mil/pubs/gdbks/risk_management.asp>.
5. Conrow, Edmund H. Effective Risk Management: Some Keys to Success. 2nd ed. Reston, VA: American Institute of Aeronautics and Astronautics, 1 June 2003.
6. Conrow, Edmund H. "Risk Management for Systems of Systems." 2004 Systems and Software Technology Conference, Salt Lake City, UT, 21 Apr. 2004.
7. Boehm, Barry, A. Winsor Brown, Victor Basili, and Richard Turner. "Spiral Acquisition of Software-Intensive Systems of Systems." CrossTalk May 2004: 4-9 <www.stsc.hill.af.mil/crosstalk/2004/05/0405boehm.html>.
8. Conrow, Edmund H. "Some Long-Term Issues and Impediments Affecting Military Systems Acquisition Reform." Acquisition Review Quarterly 2.3 (Summer 1995): 199-212.


## About the Author


**Edmund H. Conrow, Ph.D.,** is a risk management consultant to government and industry with more than 20 years experience. He has helped develop much of the Department of Defense's best practices on risk management and has also served as a risk manager on a variety of programs. Conrow is the author of "Effective Risk Management: Some Keys to Success." He has doctorate degrees in both general engineering and policy analysis.

**Risk-Services.Com**
**P. O. Box 1125**
**Redondo Beach, CA 90278**
**Phone: (310) 374-7975**
**E-mail: info@risk-services.com**

# Inherent Risks in Object-Oriented Development©

Dr. Peter Hantos
*The Aerospace Corporation*

*Object orientation has been in existence since the late 1970s. During the 1990s, however, on the basis of various claims that it was a dramatic, new software engineering approach, object-oriented software development became pervasive. Currently, most new software projects use object-oriented (OO) techniques to various extents. The persistence of schedule slips and cost overruns, particularly in the case of the development of large-scale, software-intensive systems, raises the need for revisiting the basics and exploring the inherent risks that OO technology might contribute to the overall risk profile of a project. In this article, Bertrand Meyer's classic OO technology concepts are mapped into Barry Boehm's Top 10 methodology-neutral software risks to illustrate potential areas of exposure. Recent developments in OO technology, such as Java, Use Cases, or the Unified Modeling Language fit well into this framework and are included as examples. The systematic approach introduced will allow project managers to better understand the cost/benefit aspects of applying OO technology, and to align their project management strategies more successfully with the organization's business goals.*

In this article, the term *object-oriented* (OO) technology refers to OO development processes and methods, object-related standards, and associated products and tools from third-party vendors. Enterprises that develop software are looking to OO as a means to achieve their strategic business objectives. They expect that OO will enable them to build complex systems of superior quality with reduced development time and costs, while providing long-term benefits such as maintainability, reusability, and extensibility.

If, in fact, OO has been in use for a relatively long period, then why is it still necessary to explore OO-specific risks? The simple answer can be found in R.L. Glass' 2002 article [1]. According to Glass, the introduction of a technology is no guarantee of effective use. Similar to OO, other technologies such as fourth-generation languages and computer-assisted software engineering tools were introduced with great fanfare, but once the technology was more thoroughly understood, the benefits turned out to be far more modest than originally claimed.

Also, OO risks are not the same as those associated with the introduction of any new technology. With respect to paradigm scope, complexity, and depth, OO has far-reaching consequences. For the project manager, the decision is not simply whether to apply OO to a particular project: The use of OO permeates all aspects of development. Based on business priorities, project managers must determine the desired penetration of OO concepts, the optimal insertion order, and whether the replacement of legacy languages and tools is justified.

## Object-Oriented Technology

In his 1995 book [2], Bertrand Meyer provides a sound overview of OO fundamentals. According to Meyer, software construction embracing OO is structured around the following concepts[1].

- **M1:** A unique way to define architecture and data structure instances.
- **M2:** Information hiding through abstraction and encapsulation.
- **M3:** Inheritance to organize related elements.
- **M4:** Polymorphism to perform operations that can automatically adapt to the type of structure they operate on.
- **M5:** Specialized analysis and design methods.
- **M6:** OO languages.
- **M7:** Environments that facilitate the creation of OO systems.
- **M8:** *Design by Contract*, a powerful technique to circumvent module boundary and interface problems.
- **M9:** Memory management that can automatically reclaim unused memory.
- **M10:** Distributed objects to facilitate the creation of powerful distributed systems.
- **M11:** Object databases to move beyond the data-type limitations of relational database management systems.

Please note that this article is not intended to be a tutorial on OO; rather, it will examine risk implications associated with all of these concepts. It is assumed that the reader is familiar with the basics.

## Risk Management

Risk management is acknowledged as a critical process of project management, and has received more and more attention since the 1980s. For example, in the Software Engineering Institute-developed[2] process improvement framework, during the transition from the Capability Maturity Model for Software® (SW-CMM®) to CMM Integration℠ (CMMI®), risk management was elevated from a recommended practice to a formal, independent process area. Nevertheless, to accommodate a broader audience, the definitions used in the following discussion are based on IEEE-STD-1540-2001 [3] and not CMMI materials.

Risk is defined as a potential problem, an event, hazard, threat, or situation with undesirable consequences. The non-deterministic nature of risk makes risk management a special challenge for the project manager. During project planning, we might be tempted to try to avoid risks altogether, but relying strictly on avoidance as a risk mitigation technique is usually not adequate. The success of a project depends primarily on the project manager's ability to manage the delicate balance between opportunity and risk. Unfortunately, when all risk goes away, so does opportunity. That is why successful project management practices include risk management, a continuous process for systematically addressing risk throughout the life cycle of a product or service.

According to IEEE-STD-1540-2001, the risk management process consists of the following activities:

1. Plan and implement risk management.
2. Manage the project risk profile[3].
3. Perform risk analysis.
4. Perform risk monitoring.
5. Perform risk treatment.
6. Evaluate risk management processes.

The focus of this article is risk identification, a critical aspect of risk analysis. Risk identification, similar to all other elements of continuous risk management, is


® Capability Maturity Model, CMM, and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
℠ CMM Integration is a service mark of Carnegie Mellon University.

not a one-time activity. Changes in the risk management context and changing management assumptions represent major risk sources, and need to be continuously monitored as well. IEEE-STD-1540-2001 does not prescribe how risks should be identified, but it suggests numerous methods, including the use of risk questionnaires or brainstorming.

A specialized example of a risk questionnaire, to be used in a Java 2 Enterprise Edition (J2EE) environment, is presented in [4]. Most risk questionnaires are the result of some sort of brainstorming effort; in most cases, the authors interviewed experienced project managers about their past projects and, after some filtering and processing, they turn the structured risk statements into questions or checklists. For an example of a systematic approach to develop a checklist, see Tony Moynihan's article [5].

Barry Boehm first published his Top 10 Software Risks in 1989 [6], and presented an updated list in his 1995 software engineering course with surprisingly few modifications that were based on feedback from the University of Southern California's Center for Software Engineering Industrial Affiliate companies. (For a published version of the second list please see [7].) Essentially all items, although sometimes named slightly differently, still represented major risk sources, and the name changes can be attributed to changes in popular terminology and not fundamental root causes.

## Identifying OO Risks

### *Consolidating Boehm's Risk Sources*

For the discussion in this article, Boehm's list of Top Ten Software Risks will be consolidated into eight risks as shown in Figure 1. First, items on the 1989 list were crosschecked with the 1995 list. Item No. 5, *gold-plating*, from the 1989 list is clearly a requirements mismatch issue[4]. Finally, on the 1995 list, for the sake of brevity, requirements mismatch has also been combined with user interface mismatch, and commercial off-the-shelf (COTS) issues with legacy software issues since they have many similarities with respect to root causes.

### *Mapping and Interpreting Meyer's OO Concepts*

The objective of the following analysis is to determine what OO concepts and practices are germane to risks viewed as significant by the software community. The key to meeting this challenge is the use of well-proven frameworks to inventory the essential attributes of OO technology and project risks. Boehm's risk identification checklist was chosen because it is well accepted in the software engineering community.

During the mapping process, we examined Boehm's consolidated risk list item by item and identified the corresponding, relevant OO concepts. The results of this mapping are summarized in Figure 2, and a detailed discussion follows in the rest of this article. The dots on Figure 2 represent a relationship between the particular risk item and the corresponding OO concept. Arrows pointing to the risks signify the influence of the selected OO concepts, while arrows pointing to the OO concepts relate to situations where the OO concepts have a risk-mitigating – rather than risk-triggering – effect.

#### Personnel Shortfalls (Risk B1)

Software development is a highly labor-intensive process, and its success depends primarily on the people in the organization. Beyond well-known organizational and political issues, several OO-specific concerns need to be explored. The most significant concerns are specialized skills and experience, and that is why all OO concepts are connected to this risk item as shown in Figure 2.

The first issue is the right balance between application domain knowledge and OO knowledge. It is difficult to find people skilled in both; hence, the collaboration between project personnel with different skill bases is critical. The second issue is the number and distribution of available people. OO knowledge is relevant for most members of the organization, although not to the same extent. In positions such as managers, architects, developers, and testers, it is important that all personnel have or acquire via training the appropriate OO skills.

For example, to avoid personnel shortfalls, the executives themselves who create, manage, or sponsor the development organization have to understand the essential elements of OO even before staffing starts for a project. While having prior OO experience is an asset for managers, the minimum requirement should be to have a certain level of OO literacy. In fact, Meyer's book, which is used in this analysis, is an excellent tool for this purpose, i.e., educating managers in OO fundamentals[5]. The seeding of all teams with OO mentors is also a good approach to distribute OO domain knowledge and to both jump-start and facilitate OO development.

Not surprisingly, most other sources that have analyzed OO migration have focused on the human dimension as well. Two of the three key items discussed in [4] deal with learning curve and training, and [4] contains further references to other authors addressing the same concern [8, 9].

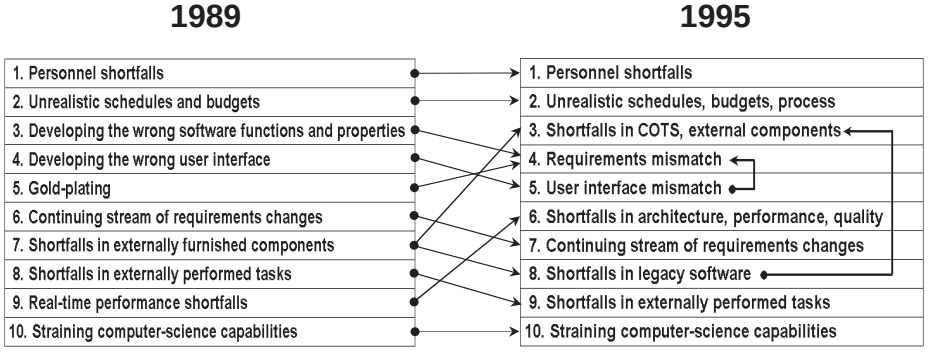
Personnel issues play an important role in the team context as well. OO requires a new way of thinking and moving away from outdated approaches like using functional decomposition for architecting systems or implementing obsolete programming constructs. For teams with a long heritage of using legacy approaches, the paradigm shift is particularly difficult. In fact, sometimes we have observed a quiet, *passive resistance* to OO methods where the people attempted to fake the usage of new methods but at the same time were continuing *business as usual*. A good example for this anomaly is writing *C*-like programs with the use of a *C++* compiler.

#### Unrealistic Schedules, Budgets, and Process (Risk B2)

Unrealistic expectations, lack of management appreciation for the necessary skills, and the difficulty of the paradigm shift will lead to unrealistic schedules. Similarly, underestimating the time and cost of necessary training would result in unrealistic schedules and budget. Nevertheless, some key OO items specifically contribute to this problem. Based on E. Flanagan's summary [10], most of the time OO projects are introduced on the following grounds:

- OO is better at organizing inherent complexity, and abstract data types make it easier to model the application.

Figure 1: *Consolidating Boehm's Top 10 Software Risks List*



(These statements are building on Concept M1, labeled Architecture and Instances.)

- OO systems are more resilient to change due to encapsulation and data hiding (per concept M2).
- OO design often results in smaller systems because of reuse, resulting in overall effort savings. This higher level of reuse in OO systems is attributed to the inheritance feature (per concept M3).
- It is easier to evolve OO systems over time because of polymorphism (per concept M4).

However, we can also learn from [5] that, particularly when OO is introduced for the first time, expectations might be exaggerated, and frequently the impact of potential costs and risks are minimized to claim maximized payback. For example, it might not be made clear to the sponsoring executives that under certain circumstances it would take several years for just the previously mentioned four benefits of OO to be fully realized. The background of this problem is two-fold. First, building class-libraries is time consuming, or, in case of purchase, they represent a major, up-front investment. Second, to achieve high return on investment, reuse must take place in a very large project or in multiple projects.

One of the side effects of the OO approach is that the design process becomes more important than it was in non-OO projects. Due to encapsulation, data hiding, and reuse, the design complexity moves out of the code space into the design space. The increased design complexity has testing consequences as well. Even if incremental integration is applied, more sophisticated integration test suites need to be created to test systems with a potentially large number of highly coupled objects.

It is also an unfortunate fact that while the OO concepts identified make system comprehension easier during analysis and design, they cause testing and debugging to become more difficult, since now all debugging methodologies and tools have to work with those abstract data types and instances. Those organizations that assume that testing OO is like testing any other software are in for a big surprise. R. Binder makes a powerful case for this argument in his article [11]. According to Binder, it is a common myth that only Black Box[6] testing is needed and OO implementation specifics are unimportant. In reality, OO code structure matters, because inheritance, encapsulation, and polymorphism present opportunities for errors that do not exist in conventional languages. Also, OO has led to new points of view and representations, and the test design techniques that extract test cases from these representations must also reflect the paradigm change.

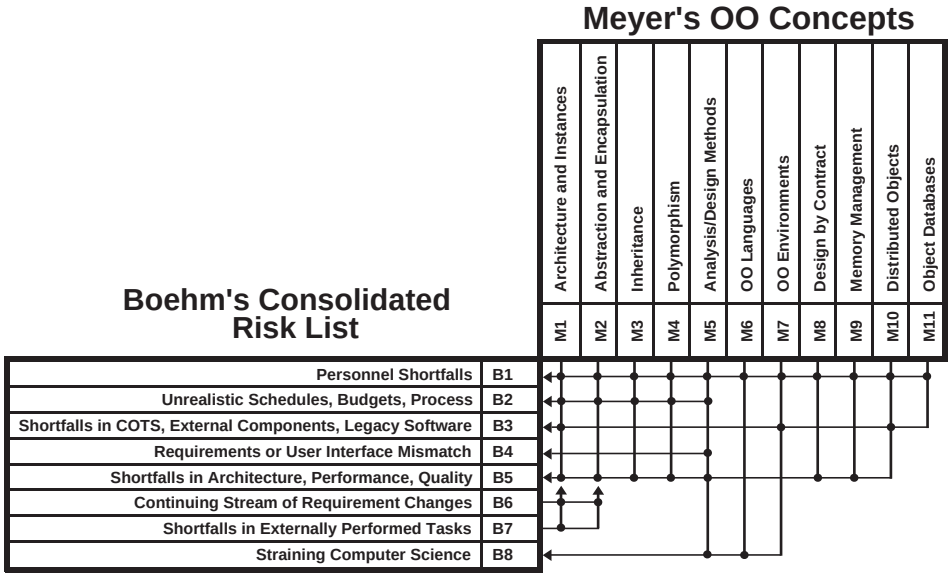



Figure 2: *Mapping Meyer's OO Concepts Into Boehm's Consolidated Risk List*

### Shortfalls in COTS, External Components, and Legacy Software (Risk B3)

Using COTS and other externally developed or legacy components in OO presents particular difficulties for structural comprehension and architectural design. These external components, their architecture, interfaces, and documentation are not necessarily consistent with the class and object architecture, communication mechanisms, and view models of the system being developed.

A particular OO problem in this area is the interface of Object Database implementations with traditional Relational Database management systems. The problem may deepen in situations where multiple new technologies merge, for example, in the use of Java-specific object-oriented COTS products (Enterprise Java Beans, Java Message Service, etc.) to develop application services on standard IBM, Sun, and Oracle platforms.

### Requirements or User Interface Mismatch (Risk B4)

The OO source of risk is the fact that use cases are used almost exclusively to develop requirements in OO systems. However, use cases only capture functional requirements so additional process steps need to be included to develop and implement quality-related[7], non-functional requirements. An interesting source of Graphical User Interface mismatch is that the Use Case methodology, though well suited for capturing the dynamism of changing screens, is inappropriate for representing screen details.

### Shortfalls in Architecture, Performance, and Quality (Risk B5)

This is the area where OO approaches present a controversial impact. Data abstraction, encapsulation, polymorphism, and the use of distributed objects, while increasing architectural clarity, all come with a price: substantial overhead due to the introduced layers of indirection. Unless the system is carefully architected and sound performance engineering practices [12] are implemented from the beginning, satisfying both performance and quality objectives becomes difficult. All of these issues boil down to the earlier mentioned design challenge. Particularly in the case of real-time applications, the system architect must carefully determine the optimal system cohesion. Most real-time performance issues can be resolved if you are willing to suffer increased coupling and the consequent loss of flexibility.

Another sensitive part of OO systems is memory management in general and the implementation of garbage collection in particular. Garbage collection is an integral part of most OO run-time environments. It is a popular technique to ensure that memory blocks that were dynamically allocated by the programmer are released and returned to the free memory pool when they are no longer needed. A typical OO application of this feature is the dynamic creation and destruction of objects. The problem is that in conventional systems, the execution of the main process needs to be interrupted while the garbage collector does its job. This ran-

domly invoked process with variable durations disrupts the real-time behavior of the system.

There are two different approaches to the mitigation of this risk. In the case of real-time OO systems, prudent programming practice should include explicit object creation and destruction to eliminate the dependency on garbage collection. Another solution is the implementation of the garbage collector via multithreading. However, multithreading is a difficult, advanced concept that itself can be the source of numerous risks. For a complete discussion of multithreading implementation pitfalls in Java, see [13].

Finally, a common, OO-related shortfall of architecture pertains to reuse. Most software development organizations moved to OO because engineering managers believed that it would lead to significant reuse. Unfortunately, as the authors of [14] point out, without an explicit reuse agenda and a systematic, reuse-directed software process, most of these OO efforts did not lead to successful, large-scale reuse. Ironically, in some other situations, even the presence of a reuse-driven agenda (platform-based product line development) is no guarantee of success if *reuse* becomes a slogan and senior management expectations are mishandled. In a product line, the participating products share (reuse) architecture and common components, and the implementation of an effective, strategic reuse process becomes a key enabler in achieving low-cost and high-quality products in a fast, efficient, and predictable way [15].

As discussed earlier, OO promises a high level of reuse via the inheritance feature and the use of class libraries. Nevertheless, OO's practical reuse is not as supportive of the described strategic reuse initiatives as one might like to see, and even the full and uncompromising implementation of OO does not guarantee the satisfaction of any aspects of the mentioned, reuse-centered corporate architecture initiatives.

#### Continuing Stream of Requirement Changes (Risk B6)

This risk is caused by customer behavior, and the use of OO is not a contributing factor. On the contrary, as it was pointed out in Risk B2, OO architectural considerations, encapsulation, and data hiding increase the developed system's resiliency to requirements volatility.

#### Shortfalls in Externally Performed Tasks (Risk B7)

Risk B7 is caused by contractor behavior, and the use of OO does not play any role. Nevertheless, similar to B6, the presence of M1 and M2 OO concepts is an excellent mitigating factor when these kinds of problems arise.

#### Straining Computer-Science Capabilities (Risk B8)

The appeal of the concepts M1-M4 (see Figure 2), which are theoretical in nature, inspires system architects to use OO in designing complex systems. Concepts M5-M7 are related to implementation, and their role is to enable and facilitate using the theoretical concepts. This risk item refers to the persistent tension between the theoretical concepts and their implementation, and the delicate balance that must be maintained among programming languages, developing environments, and analysis/design methods.

The viability and feasibility of all these elements have to be continually verified against the developed system's architecture. A recent example is the introduction of a promising new programming technique called Aspect-Oriented Programming (AOP). According to Gregor Kiczales, one of the principal developers of AOP, integrating AOP with OO development environments is difficult [16]. A standard development environment would have facilities for structure browsing, smart editing, refactoring, building, testing, and debugging, but it does not have a way to represent and directly manipulate AOP-specific constructs.

## Summary

A systematic approach was presented to identify risks in OO development. The fundamental concepts of OO were introduced and matched against a well-known, methodology-neutral list of software risks. This dissection of OO concepts allows project managers to more completely understand the cost/benefit aspects of applying OO, and to align their project management strategies better with the organization's business goals.◆

## Acknowledgements

This work would not have been possible without assistance from the following people and organizations:

- Reviewers: Richard J. Adams, Sergio Alvarado, Suellen Eslinger, and Joanne Tagami all with The Aerospace Corporation, and Scott A. Whitmire at ODS Software, Inc.
- Sponsor: Michael Zambrana, U.S. Air Force Space and Missile Center.
- Funding Source: Mission-Oriented Investigation and Experimentation Research Program, Software Acquisition Task.

A version of this article was presented at the 2004 Pacific Northwest Software Quality Conference (2004 PNSQC).

## References

1. Glass, R.L. "The Naturalness of Object Orientation: Beating a Dead Horse?" IEEE Software May/June 2002.
2. Meyer, B. Object Success. Prentice Hall PTR, 1995.
3. The Institute of Electrical and Electronics Engineers. IEEE-STD-1540-2001 – Standard for Software Life Cycle Processes-Risk Management. New York: IEEE, 2001.
4. Merson, P. "Managing J2EE Risks." Software Development July 2004.
5. Moynihan, T. "How Experienced Project Managers Assess Risk." IEEE Software, May/June 1997.
6. Boehm, B. IEEE Tutorial on Software Risk Management. IEEE Computer Society Press, 1989.
7. Boehm, B. "Software Risk Management: Overview and Recent Developments." 17th International Forum on COCOMO and Software Cost Modeling. Los Angeles, CA, Oct. 2002.
8. Fichman, R., and C. Kemerer. "The Assimilation of Software Process Innovations: An Organizational Learning Perspective." Management Science, 1997.
9. Feiman, J. Migrating Developers to Java: Is It Worth the Cost and Risks? Stanford, CT: Gartner, 2000.
10. Flanagan, E.B. "Risky Business." C++ Report Mar.-Apr. 1995.
11. Binder, R.V. "Object-Oriented Testing: Myth and Reality." Object Magazine May 1995.
12. Smith, C.U. Performance Engineering of Software Systems. Addison-Wesley, 1990.
13. Sandén, B. "Coping with Java Threads." IEEE Computer Apr. 2004.
14. Jacobson, I., et al. Software Reuse. ACM Press, 1997.
15. Northrop, L.M. "A Practical Look at Software Product Lines." CASCON 2003, Ontario, CA, Oct. 2003.
16. Kiczales, G., and M. Kersten. "Show Me the Structure." Software Development Apr. 2000.

## Notes

1. Please note that the M1-M11 numbering of concepts did not originate from Meyer; it was introduced by the author to facilitate the mapping process.
2. The Software Engineering Institute is

a federally funded research and development organization at Carnegie Mellon University, Pittsburgh, Pa.
3. Risk profile: A chronological record of a risk's current and historical state information [3].
4. Gold-plating is a popular software management term for implementing features by software engineers that go beyond the scope of actual requirements.
5. Consider the book's subtitle: "A manager's guide to object orientation, its impact on the corporation and its use for reengineering the software process."
6. Black Box testing targets externally observable behavior that is produced from a given input, without using any implementation information.
7. Quality in short is fitness for purpose, the degree to which a system accomplishes its designated functions within constraint. It includes all the -ities, e.g., availability, reliability, security, safety, etc.

## About the Author

**Peter Hantos, Ph.D.,** is currently a senior engineering specialist in the Software Acquisition and Process Office of the Software Engineering subdivision at The Aerospace Corporation. He has more than 30 years of experience as a professor, researcher, software engineer, and manager. Previously as principal scientist at the Xerox Corporate Engineering Center, Hantos developed corporate-wide engineering processes for software-intensive systems, and as department manager, he directed all aspects of quality for several laser printer product lines. He is author of numerous technical papers and U.S. and international conference presentations. Hantos is a member of the Association for Computing Machinery and senior member of the Institute of Electrical and Electronics Engineers. He has a Master of Science and doctorate degree in electrical engineering from the Budapest Institute of Technology, Hungary.

**The Aerospace Corporation**
**P.O. Box 92957 – M1/112**
**El Segundo, CA 90009-2957**
**Phone: (310) 336-1802**
**Fax: (310) 563-1160**
**E-mail: peter.hantos@aero.org**

## Web Sites

### Risk Management
www.acq.osd.mil/io/se/risk_management/index.htm
This is the Department of Defense (DoD) risk management Web site. The Systems Engineering group within the interoperability organization formed a working group of representatives from the services and other DoD agencies involved in systems acquisition to assist in the evaluation of the DoD's approach to risk management. The group will continue to provide a forum that provides program managers with the latest tools and advice on managing risk.

### Software Technology Support Center
www.stsc.hill.af.mil
The Software Technology Support Center is an Air Force organization established to help other U.S. government organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and to accurately predict the cost and schedule of their delivery.

### Software Program Managers Network
www.spmn.com
The Software Program Managers Network (SPMN) is sponsored by the deputy under secretary of defense for Science and Technology, Software Intensive Systems Directorate. It seeks out proven industry and government software best practices and conveys them to managers of large-scale DoD software-intensive acquisition programs. SPMN provides consulting, on-site program assessments, project risk assessments, software tools, and hands-on training.

### Software Engineering Institute
www.sei.cmu.edu
The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the Department of Defense to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. SEI helps organizations and individuals improve their software engineering management practices. The site features complete information on models the SEI is currently involved in developing, expanding, or maintaining, including the Capability Maturity Model® Integration, Capability Maturity Model® for Software, Software Acquisition Capability Maturity Model®, Systems Engineering Capability Maturity Model®, and more.

### Project Management Institute
www.pmi.org
The Project Management Institute (PMI) claims to be the world's leading not-for-profit project management professional association. PMI provides global leadership in the development of standards for the practice of the project management profession throughout the world.

### The Software Productivity Consortium
www.software.org
The Software Productivity Consortium is a nonprofit partnership of industry, government, and academia. It develops processes, methods, tools, and supporting services to help members and affiliates build high-quality, component-based systems, and continuously advance their systems and software engineering maturity pursuant to the guidelines of all of the major process and quality frameworks. Based on the members' collective needs, its technical program builds on current best practices and information technologies to create project-ready processes, methods, training, tools, and supporting services for systems and software development.

### INCOSE
www.incose.org
The International Council on Systems Engineering (INCOSE) was formed to develop, nurture, and enhance the interdisciplinary approach to enable the realization of successful systems. INCOSE works with industry, academia, and government in these ways: provides a focal point for disseminating systems engineering knowledge, promotes collaboration in systems engineering education and research, assures the establishment of professional standards for integrity in the practice of systems engineering, and encourages governmental and industrial support for research and educational programs to improve the systems engineering process and its practices.

# Software Risk Management From a System Perspective

George Holt
*AdaRose Inc.*

*Software development can be fraught with frustration. Too often, we treat hardware risks and software risks as separate entities. Staying focused on the basics of risk management at the system level, from the get-go, is an essential part of minimizing risks and ensuring the success of even the most challenging and complex development projects. This article stresses the importance of managing risk from a system perspective by providing concrete examples of how one company applied the fundamentals of risk management to a military tactical system developed under less than ideal conditions.*

Developing software can be challenging and rewarding but seldom easy. Developing software with a floating hardware baseline can be quite difficult. Add to this the commensurate development of test tools, simulators, and emulators by third parties, and then place schedule and cost constraints on the entire project, and you challenge even the best and the brightest.

Although each project entails unique demands, challenges, and problems, if we fail to predict and prevent risks from a system perspective it can lead to costly delays, increased stress on team members, a lesser product – even project failure.

This article stresses the importance of managing risk from a system perspective. It provides concrete examples of how one company, AdaRose Inc., applied the fundamentals of risk management to a military tactical system developed under less than ideal conditions such as those described above.

## The Task at Hand

AdaRose engineers had prior experience developing software that resulted in the first tactical weapon system to run on a common PC architecture using a commercial operating system. The current task was to port this software to a new hardware architecture (still PC-based) that incorporated three single-board computers providing navigation, command-and-control, situational awareness, and real-time diagnostics.

Along with AdaRose, the Integrated Product Team (IPT) consisted of a major defense contractor, a small business hardware manufacturer, the Army end user, Army research and development specialists, and the Army product manager. All members of the IPT were experienced professionals with in-depth knowledge of the weapon system from both a functional and operational perspective. The software consisted of 230,000 lines of Ada code with specialized modules and drivers written in other high order languages.

## The Solution

Although each project has its own requirements, the fundamentals of effective risk management at the system level remain the same. By identifying risks and developing solutions before and during the development process, you maximize the team's efficiency and the quality of the finished product.

*"... the fundamentals of effective risk management at the system level remain the same. By identifying risks and developing solutions before and during the development process, you maximize the team's efficiency and the quality of the finished product."*

I would like to start off by referring the reader to one of my earlier articles, "Risk Management Fundamentals in Software Development" published in the August 2000 issue of CrossTalk [1]. It describes how to implement an effective software risk management program. The fundamentals in that article can be applied, at the system level, to this military tactical system developed under less than ideal conditions.

## Identifying the Risks

From the get-go, we were informed that this project would have significant risk drivers, i.e., (1) it would have cost and schedule constraints, (2) it would require software development before the hardware was built, and (3) tools such as lab simulators and emulators would have to be developed commensurate with the tactical software development. What was initially perceived as a straightforward port of software to a new hardware environment turned out to be a nontrivial undertaking.

## System Level Risks

The challenge on this project soon became evident. On the one hand, software could not wait for completion of hardware due to the schedule constraint. This required us to proceed with software design and development without access to a hardware target platform. Additionally, there was a requirement for building simulators and emulators for both development and testing. However, some of these tools, being built by Army engineers, would need to be certified before use and certification required running on hardware and software that was still under development.

These parallel development efforts would require a unique approach to development and risk management. At the macro or program level, we identified the following risk drivers.

1. **Schedule:** The schedule would be constrained and success-oriented, and the highest priority was placed on meeting schedule to allow for early fielding of the system. Time- and labor-intensive tasks such as documentation might have to be deferred until late in the schedule. In addition, many tasks that would normally be done sequentially would have to be done in parallel.
2. **Funding:** Limited funding was available for the software portion of the program. AdaRose would plan to make maximum use of available funding by multiple tasking of full-time engineers and by utilizing part-time

labor for engineering support elements such as configuration management (CM), quality assurance (QA), lab technicians, and network maintainers.

3. **Technical:** A number of engineering challenges were evident. The situational awareness (SA) computer had to be integrated to ensure that any SA failure would not impact the primary mission computer. Also, two third-party products – a radar measuring unit and a tactical communication module – needed to be integrated. AdaRose engineers had past familiarity with the tactical software, as well as prior experience with integrating situational awareness functionality and third-party products. Therefore, technical risk, although evident, was placed third in priority, as it did not appear to be a *showstopper* for the program.

## Risk Scenarios

At the start of the program, we developed a number of risk scenarios to determine those events or trigger points we would have to watch, to warn us if and when the risk became imminent. Even though technical was not a serious risk driver, our No. 1 risk scenario involved the potential that the hardware, still in the design and development stage, would be substantially different from the specifications we were working from. If so, it could entail software rework and impact cost and schedule.

Our No. 1 concern was the communication interface between the tactical application and the inertial measurement/navigational unit. In the legacy system, this had been a straightforward Direct Memory Access (DMA) interface. Any change here was very risky because this unit was at the heart of the system and failure here meant the system could be dead in the water. The trigger point we watched for in this risk scenario was any change to that interface – and sure enough it occurred as the project evolved.

Due to hardware limitations on the tactical single-board computer, DMA could not be supported and the communication between the tactical application and the navigational unit had to be changed from DMA to an interrupt-driven serial connection. This, in turn, drove additional requirements to develop four new drivers to replace a single generic driver contained in the old architecture. This risk was mitigated somewhat by the fact that AdaRose engineers had prior formal training in developing software drivers for this operating system.

## Controlling the Risks

As a baseline to accommodate top-level program visibility, AdaRose normally uses the typical Stair Step development process consisting of (1) requirements analysis (RA), (2) design, (3) code and unit test (CUT), (4) system level integration and test (SIT), and (5) formal qualification test (FQT). Then, depending on the type of software to be developed (e.g., new development, re-host, block update, prototype, etc.) and the constraints placed on the program (e.g., cost, schedule, technical), this baseline is modified/augmented for best program performance.

For this program, we decided to modify the baseline process with a spiral development approach to obtain maximum productivity from our developers and to mitigate major risk areas. At any point in time, programmers would be coding and unit testing in some areas while requirements analysis or design would be proceeding in others. We could also move out in those areas where the software was not yet dependent on hardware availability. For example, we decided, early on, to develop, application-specific, software simulators and communication protocol simulators to test the software – especially in those technical areas where rapid prototyping for proof-of-concept or early risk mitigation was warranted. The threads we used, throughout the development effort, to maintain coherency became known as *feature sets*. We found these to be invaluable risk mitigators.

> *"For this program, we decided to modify the baseline process with a spiral development approach to obtain maximum productivity from our developers and to mitigate major risk areas."*

## Feature Sets

A feature set is a block of executable software that contains predefined features/ requirements that make up a subset of the entire program/application. A feature set can consist of nothing more than a rapid prototype to determine proof-of-concept, or a fully integrated and tested baseline. The purpose of feature sets is (1) to put before the user periodic drops of executable code to gain early concurrence and feedback of the included features/requirements; (2) to conduct early-on testing to reduce program risk and provide relatively *bug-free* software prior to entering FQT; and (3) to keep the development effort moving by allowing developers to move forward on those sets of features that are not dependent on other events, such as delivery of target hardware, special tools, or third-party products.

Ideally, as the program progresses and the software matures, periodic drops of feature sets would consist of the most current feature set along with all previous sets until such time that the final set is incorporated and the application is ready to enter FQT. Most of the early feature sets were tested using the developed software simulators.

The other *system level* risk mitigators that we used and that were described in my earlier articles on risk management [1, 2] are in the following sections.

## Integrated Product Teams

Forming IPTs is another valuable approach to containing costs and reducing risks, especially those that might effect scheduling. The IPT facilitates problem solving, enables the team to rapidly respond to changing requirements, and prompts everyone to work on schedule.

## Prototyping

Exploratory prototyping is an excellent risk mitigator if project requirements are ill-defined or likely to change before project completion. In addition, exploratory prototyping is an excellent way to clarify requirements, identify desirable features of the target system, and promote the discussion of alternative solutions.

Prototyping should answer two questions that are fundamental to software development and risk management: "Is the concept sound?" and "Is it worth proceeding further?" If the answer is not a clear *yes*, you may be setting yourself up for failure. More importantly, without this insight, you will give the customer a false sense of what can be accomplished. It is better to know this up front. Sometimes the most important risk management action you will take is to ask these fundamental questions.

As an example, on this project we needed to determine whether or not a viable software solution could be found

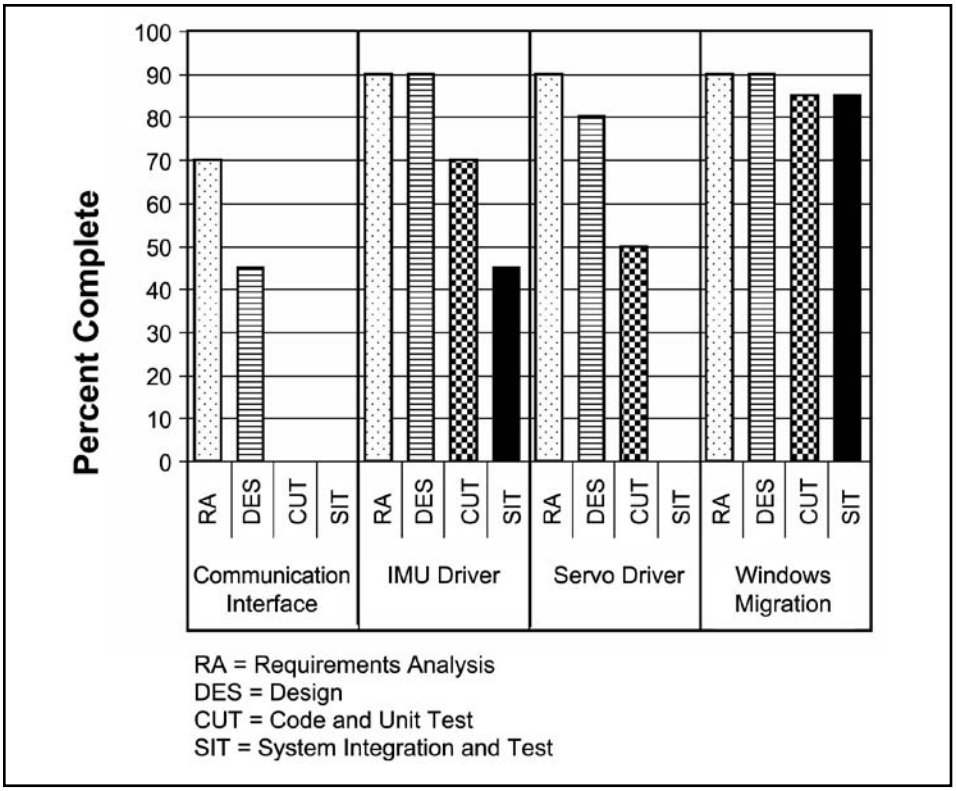



Figure 1: *Example of a Weekly Metric*

to replace the aging analog tachometers that controlled the rate of movement of the weapon system. We discovered that rate data was obtainable from the inertial measurement/navigational unit. We then proceeded to develop the prototype algorithms that substituted this rate data for the data from the tachometers. The next step was to *prove the concept*. This required just enough recoding to make it work on the existing system hardware. This was successful and as a result we were able to mitigate this risk early in the program.

If the answers and the risks are satisfactory in the exploratory prototyping phase, you can move on to evolutionary prototyping, which offers several benefits. It enables your team to quickly and efficiently build on proven aspects of the software. As a result, the core of the software's foundation is tested and proven early in the project, significantly reducing exposure to unknowns. It is an important contributor to *feature sets*.

## Process Improvement

Improving processes should be ongoing throughout the project. For example, this project required a dual display mode on the operator's console. Rather than hold up development, while waiting on hardware to arrive, we did the necessary design and coding and used a dual monitor graphics card to test out and *prove* the design.

It is important to continually ask, "Is there a better way to get the job done?" Improving the way you do things cannot be done in a vacuum – communication at all levels is critical. Participate with your customers in IPTs and system management teams. In addition, be sure to meet with the teams' engineers on a regular basis for focused, but informal, discussions. While these meetings are exceptionally valuable, guard against extended meetings that cut into your teams' work time.

One alternative to lengthy meetings is to develop and distribute weekly status reports. These give each member insight into the progress of the entire project and a clear view of the big picture. Remember that you can have the best processes in place and still fail miserably in software development. A motivated, goal-oriented, and knowledgeable workforce will succeed even when the process is lacking. An example of one metric we used on a weekly basis is displayed in Figure 1.

## Percent Complete

This metric provided top-level insight to the stage of development across blocks of functional requirements. We have shown here only four of the 13 major functional areas. Note that work in the communication interface area had not yet entered the code-and-unit test phase due to unavailability of hardware being developed by a third party. However, Windows migration was well ahead of the curve because it was not hardware dependent.

Also included in the weekly reports were more detailed descriptions of the major risk areas, for example see Figure 2.

Risk rankings were continuously re-evaluated and reprioritized throughout the program. As higher priority risks were worked off, others would move up to take their place. Risk mitigation became a dynamic real-time process.

Figure 2: *Example of Detailed Descriptions of a Major Risk Area*

**IMU DRIVER**

Complete, integrate, and deliver with Feature Set #7a.
Test the integrated build at the system level.

**Risk Priority Ranking = 2**
Schedule: Moderate/High
Technical: Moderate
Cost: Low/Moderate

**Risk Mitigation:** Develop dedicated IMU driver and modify application code IAW established DRU-H driver development plan. Apply best people. Utilize appropriate tools (SoftICE DDK). Develop software communication simulator. Use logic analyzer and oscilloscope for low-level debugging.

**Action Officer(s)**: Primary – Tom; SW engineering – Frank and Ilya.
**Suspense:** IAW driver development plan, until removed from risk list.

## Third Parties: A Mixed Blessing

If a product does fail, it is common for many developers to blame the project's failure on third parties. In some cases they are correct. At times you will have no choice but to elicit their help. The key is to minimize how much you depend on them.

Any time you rely on a product or service from someone outside of your group your risk of failure or delay increases. Your team may do everything right, but if a crucial third party does not, your work may be

in vain. To illustrate this, consider the risks you assume by depending on three crucial components of your project from start to finish. Assume each product has an 80 percent chance of arriving on time and fully functional. The probability of success for all three combined is not 80 percent, it is 0.80 x 0.80 x 0.80 or 51 percent. In other words, your project now has only a 50-50 chance of success. Do not assume that third parties will have the same priorities that you have. Use daily communication with them to keep them in the loop and make them a part of your team.

## System-Level Cohesion

Needless to say, software cannot be developed in a vacuum. In the ideal software world, we would hope to have qualified hardware, emulators/simulators, and all design and interface documents delivered at project start. But that is not realistic, especially with military tactical systems. We find that software and hardware are an integral, non-separable entity. Quite often participants in *system* development efforts will *finger point* and blame the other guy for lack of progress.

On this project, we all realized the many challenges and knew that a successful outcome depended on a strong team effort. As a result, we witnessed almost daily instances of engineers supporting each other – often putting aside their own work to help move forward a higher priority effort. We saw a close working relationship develop between our software developers and the hardware developer. AdaRose engineers quite often diagnosed hardware anomalies and provided workable solutions. At the IPT level, all were well aware of the risks and a *helping hand* rather than a *pushing hand* was the norm.

## Results

As of this writing, the project is midway through formal qualification test. The schedule is still paramount but software development was able to proceed in advance of hardware availability by identifying and mitigating those critical risk areas that could be worked off early. We did this, up front, through rapid prototyping and by providing feature sets to show proof-of-concept and provide executable code to qualify the hardware and help certify the simulation/emulation tools. This required a tailoring of our process and maintaining a viable and dynamic risk management program at the system level.

## Summary

Software development will always include risks, but none are insurmountable if you are prepared to face them at the start. Risk management is an excellent way to prepare for daily challenges. Risk management must not only be implemented but continually reassessed throughout the life of the project. Do not blindly follow any particular process but do tailor your process to the job at hand.

A viable risk management plan can mean the difference between success and failure. It should, above all else, be flexible and encourage initiative. Remember to always look ahead, use rapid prototyping if necessary, develop simulators if necessary, follow a defined program to minimize and manage risks, use a good set of metrics, keep the customer in the loop, and always follow the fundamentals of sound application development. Following this risk management approach will not guarantee excellent software development, but over time it will certainly contribute to your success.◆

## References


1. Holt, George. "Risk Management Fundamentals in Software Development." CROSSTALK, Aug. 2000: 12-14 <www.stsc.hill.af.mil/crosstalk/2000/08/holt.html>.
2. Holt, George. "Software Risk Management – The Practical Approach." Software Tech News 2.2 <www.softwaretechnews.com/technews 2-2/practical.html>.


## About the Author


**George Holt** is president and chief executive officer of AdaRose Inc. He has a wealth of program management experience primarily with military tactical systems. AdaRose recently re-hosted 230,000 lines of Army tactical software, written in Ada, to a digital control unit containing three single-board computers, providing navigation, artillery fire control, situational awareness, and a prognostic/diagnostic capability. Holt is the author of many technical publications and co-author of "Strategy: A Reader."

**AdaRose Inc.**
**430 Marrett RD**
**Lexington, MA 02421**
**Phone: (802) 728-9448**
**Fax: (781) 274-7359**
**E-mail: holt.adarose@verizon.net**


## Coming Events

**March 5-12**
*IEEE Aerospace Conference*
Big Sky, MT
www.aeroconf.org

**March 7-10**
*SEPG 2005*
Seattle, WA
www.sei.cmu.edu/sepg

**March 15-16**
*Dayton Information Security Conference '05*
Dayton, OH
www.gdita.org/inc/eventdetail.asp?eventID=340

**April 4-6**
*DTIC Annual Users Meeting and Training Conference*
Alexandria, VA
www.dtic.mil/dtic/annualconf

**April 5-7**
*Federal Office Systems Exposition (FOSE) 2005*
Washington, DC
www.fose.com

**April 18-21**
*2005 Systems and Software Technology Conference*
Salt Lake City, UT
www.stc-online.org

**May 2-6**
*Practical Software Quality and Testing (PSQT) 2005*
Las Vegas, NV
www.qualityconferences.com

**May 8-12**
*Nano Science and Technology Institute 2005 Conference*
Anaheim, CA
www.nanotech2005.com/

# Managing Acquisition Risk by Applying Proven Best Practices

Mike Evans and Corinne Segura
*American Systems Corporation Inc.*

Frank Doherty
*U.S. Navy*

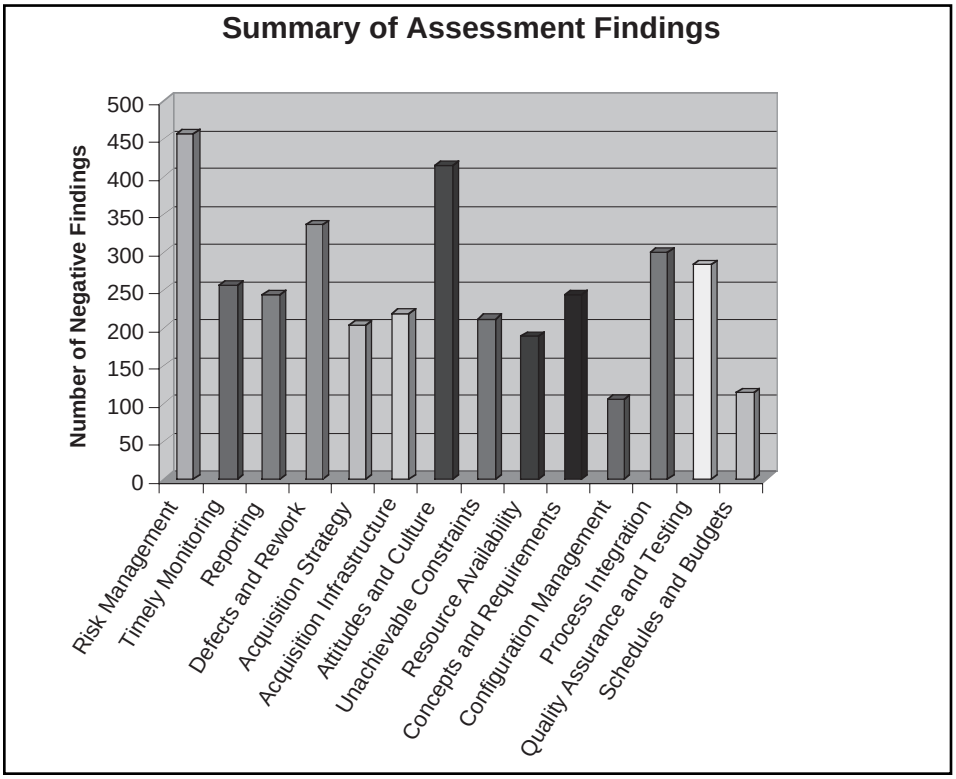
*Data analysis from recent acquisition program assessments has identified common characteristics of successful programs and supporting organizations. First and foremost, organizations with successful acquisition processes must embrace risk management throughout the entire product life cycle. While risk management is ingrained within their culture, these organizations take active measures to sustain effective implementation across programs by routinely conducting assessments to maintain currency, applying proven best practices to address specific risks, and using historical* lessons learned *to improve future performance. These assessment results also revealed characteristics of* unsuccessful *programs, primarily a lack of understanding and distinction between acquisition and development processes. This confusion resulted in an increase in interface issues as well as observable impacts on product cost, schedule, and quality. As a result of their analysis, the authors conclude that successful acquisition risk management is based on: (1) providing educated leadership and a supportive organizational culture, (2) adapting proven best practices in response to specific circumstances, and (3) emphasizing the program environment rather than process maturity.*

During 2003, the American Systems Corporation (ASC) conducted nine program assessments of commercial and government organizations. These assessments evaluated 50 individual acquisition projects that were components of larger programs. Approximately half were acquisition programs with the remainder being programs to develop a product or provide a service.

The ASC assessment approach used a series of automated evaluation tools based on the revised Department of Defense (DoD) 5000 series of instructions, acquisition process models, a best practices-based model, evaluation criteria similar to the current Class C Standard Capability Maturity Model® Integration (CMMI®) Assessment Method for Process Improvement-based model, and various specialized evaluation tools.

One of the major assessments was a program under a major Navy acquisition command responsible for acquiring hardware and software for afloat platforms. The ASC assessed the overall acquisition performance and associated risks within this program office by utilizing the assessment process described above in conjunction with the ASC Gap Analysis Profiling (GAP) tool.

Figure 1: *Observation Summary*




The ASC employs a consistent and repeatable process to conduct and analyze results for all assessments. The process begins with data collection and is accomplished by using a variety of questionnaires depending on the assessment model. Assessors conduct interviews, review documentation, and record their observations and document issues, which they then analyze manually using the automated GAP analysis tool.

Outputs include a matrix of risks associated with specific business processes that are weighted and sorted by various criteria, and a histogram that represents a compilation of all data points that identify high-risk areas and prioritize areas for process improvement. The assessors also correlate their observations and issues against proven best practices such as the Software Program Managers Network (SPMN) 16 Point Plan, CMMI criteria, DoD 5000 requirements, Operational Test readiness criteria, or customized evaluation points based on customer needs. The results are then documented in a final report with a consistent format and saved as a series of program-specific reports.

## Assessment Observations

When we compiled all of the 2003 assessment results (government and commercial), we observed an interesting anomaly. The initial results of a commercial assessment composed of a series of 20 programs identified two areas of strengths: architecture development and interface development. Further analysis indicated that these programs had the largest cost and schedule growth of any in the information technology portfolio. This observation was inconsistent with what was originally expected.

When we reanalyzed the results of our initial assessment, we identified several factors that explained these anomalies:

- Management had an unrealistic *can-do-at-all-cost* attitude that prevented an objective assessment of their actual capabilities to contain risk and control rework. This attitude prevented them from using available processes correctly, and it prevailed despite the fact that the technology being used in the programs appeared to be adequate. Such a *can-do* attitude introduces the risk that the program will continue on an unproductive path despite irrefutable evidence that it will not progress to the desired end state. For example, with this attitude, management would possibly dedicate more people and dollars to a problem that is related to ineffective processes rather than address the processes.
- Management failed to identify and remove defects that reduced product quality. They failed to manage and mitigate risks, which negatively affected cost, schedule, performance, and services provided.
- For many of these programs, management failed to distinguish adequately between development and acquisition practices.

When we reanalyzed the more than 900 observations that were collected during the initial assessments, we included a new categorization scheme that focused on the program environment. As shown in the histogram in Figure 1, the most significant issues regarding cost and schedule growth, which seemed to be more significant than process-related issues, were the attitude and culture of management and project personnel, and the project's ability to effectively manage risk. In addition, issues related to productivity and performing to a plan were far more prevalent than issues related to estimating cost or projecting schedules. Finally, program team members seemed to be more aware of process integration factors than specific shortfalls in individual processes. We concluded that, in terms of probability of success, this program was being affected more by the program environment than by process shortfalls.

During our reassessment, we observed that the client's employees consistently described practices in the wrong context. For example, individuals in acquisition organizations described the practices they were using to control development baselines, the methods they planned to use to develop the software architecture, or how they planned to use testing to resolve product quality issues.

When we evaluated this confusion of practices, we determined that there was extensive definition of *development* best practices in the form of initiatives such as the 16 Point Plan, Practical Software and Systems Measurement (PSM), several Office of the Secretary of Defense (OSD) studies, and initiatives from the Software Engineering Institute and the Data and Analysis Center for Software. However, there were fewer initiatives related to proven best practices in the area of *acquisition*, with many of these practices blending into overarching models such as CMMI.

We discovered that in many organizations we assessed, program team members often confused development practices with practices more relevant to acquisition. In an acquisition environment, practices related to development can be useful, but they must be adapted to the specific requirements of receiving a product rather than building it, and this adaptation does not always occur.

Figure 2 illustrates various practices that must be adapted to work within the larger organization and to fill a specific role within the context of the overall program. As Tim Lister put it at the 1996 Software Technology Conference, "Could it be that adaptation of process is 90 percent of the problem, and the common processes are marginal?" [1]. This quote provides evidence that practitioners within the industry are concerned about successful implementation of best practices in a project environment.

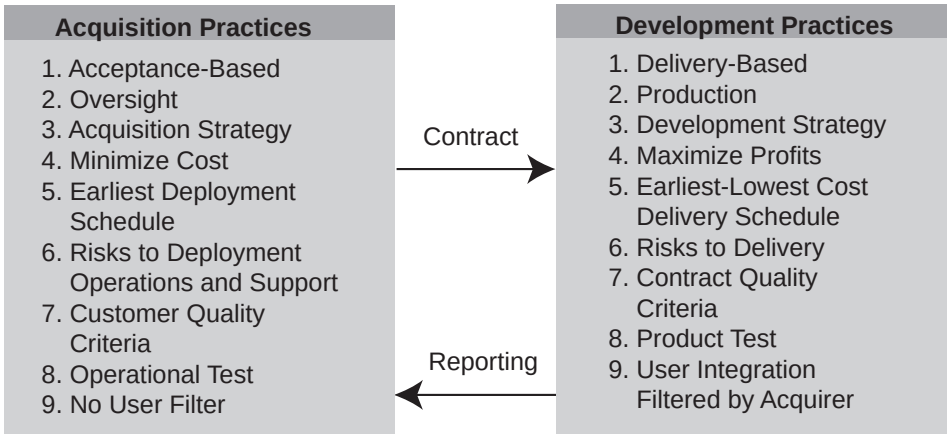



Figure 2: *Organizational Considerations*

As Figure 2 illustrates, similar practices must be substantially adapted to meet the differing needs of the acquirer and developer.

To facilitate effective adaptation of common practices, we developed the *Issues Grid* (Figure 3) to distinguish between acquirer and supplier functions as they relate to nine common issue areas. As the Issues Grid highlights, the risks that arise within these areas are specific to the role the organization plays in the project, and the response to these risks is driven by dif-

Figure 3: *Issues Grid*

| Issues Grid | | | | |
|---|---|---|---|---|
| **Supplier Issues** | | **Common Issues** | | **Acquirer Issues** |
| Delivery Integrity | | Risk Management | | Deployment and Support Integrity |
| Product Acceptance | | Requirements | | Operational Effectiveness and Support |
| Project Breakdown | | Baseline Control | | Product Integrity |
| Inflexible Project Change Delays Tradeoff Constraints | | Contracting | | Cost, Schedule and Product Exposure |
| Product Integrity, Acceptance and Delivery | | Defect Identification and Tracking | | Product Reliability, Performance and Operational Integrity |
| Product Demonstration Project Completion | | Interfaces and Interoperability | | Deployment, Operational Integrity and Support |
| Cost Overrun, Late Delivery, Profit, Impacts, and Excessive Constraints, Cost, Schedule, and Product Exposure | | Estimation and Scheduling | | Tradeoff Constraints, Change Delays, and Contingency Management, Cost Overrun, Late Delivery and Excessive Constraints |
| Constant Surprises and Project Chaos | | Metrics and Reporting | | Tradeoff Based on Incorrect Information |
| Staff Attrition | | Staff Issues | | Staff Consistency |

fering organizational motivations and commitments.

From our observations in 2003, the attitudes of management and staff appeared to be a driver in program success. Typical comments were as follows:

- "I know there's risk but the only contract type we have time to manage is FFP [firm fixed price], which shoves all risk to the contractor."
- "The review is next week. We have to wing the estimate or we won't get funded."
- "Schedule? When do you need it?"
- "I don't know what you'll find when they start using it. It'll be good enough."
- "The staff will just have to 'suck it up.' I can't afford the overtime."
- "If I tell management that, they'll fire me."

These quotes not only indicate the frustration of the various project stakeholders, but also the divergence that can exist in how management, the customer, the staff, and the users understand the motivations and commitments of different organizations and individuals. In such an environment, a program has little chance of success either because individual commitments are unrealistic or morale is so poor.

The authors have observed many times that successful implementation of any practice, whether it can be considered a best practice or not, depends more on how the practice is accepted within the program's culture and how specifically it is integrated rather than the value of the concept it provides. For example, in regard to risk management, we have observed that every organization we have assessed explicitly accepts the value of this practice. We often hear comments like, "We need to know what can impact our program early so that we can better manage it," or "Risk management is essential to our success or failure since it provides us an early warning."

However, very few of the organizations we assessed truly embrace the process: Very few managers are willing to completely report negative risks to senior management for fear of negative reaction or unwanted help. Only an organization that culturally embraces risk management would assume the posture that management needs to be aware of the potential for good and bad outcomes.

## Analysis and Conclusions

Based on our reassessment of our 2003 observations, we reached certain conclusions. First, for an acquisition program to be successful, the program must be planned and adequately staffed and resourced. It also must be consistently executed and follow acquisition strategies that are aligned with enterprise and organizational guidelines. The processes used must be documented and, most importantly, they must be adapted to the specific role of the organization using them; the culture of that organization; and the realities of staff, schedule, and resources. Additionally, those processes that are critical to acquisition success must be cultural imperatives, and they cannot outpace the skills, training, and experience of the individuals who must apply them. Finally, an acquisition organization must do more than simply define the process. A primary task must also be to identify, tailor, acquire, integrate, apply, and monitor the effectiveness of the individual practices, methods, and tools that are used to implement the process. Understanding what to do (process) is important, but understanding how to do it (practice) is critical.

Because this observation is common knowledge, the question becomes, "Why don't we deal with it?" Impediments to the implementation of a process often are not inherent to the process itself, but rather they arise from the organizational culture. The CROSSTALK article "Seven Characteristics of Dysfunctional Software Projects" [2] indicates some causes of poor organizational culture. It identifies seven specific project characteristics that preclude an organized application of effective practices to a project:

1. Unwarranted optimism and the unrealistic expectations of executive management.
2. Late decision-making.
3. Inappropriate use of the standard software process.
4. Missing or inadequately implemented program activities.
5. Lack of leadership.
6. Early declarations of victory.
7. Absence of risk management.

When these characteristics exist on an acquisition project, an attitude develops that is extremely detrimental to success. The question then arises, "If these issues are so apparent, why don't projects address them?" As indicated in [2], the two primary reasons most likely are denial and culture. Denial becomes an issue when, in the day-to-day execution of an acquisition project, an attitude develops that can be characterized this way: "The indicators of disaster are probably wrong, and we won't be impacted the way the other 12 projects were." Such an attitude can lead acquisition managers, or any manager for that matter, to do risky things.

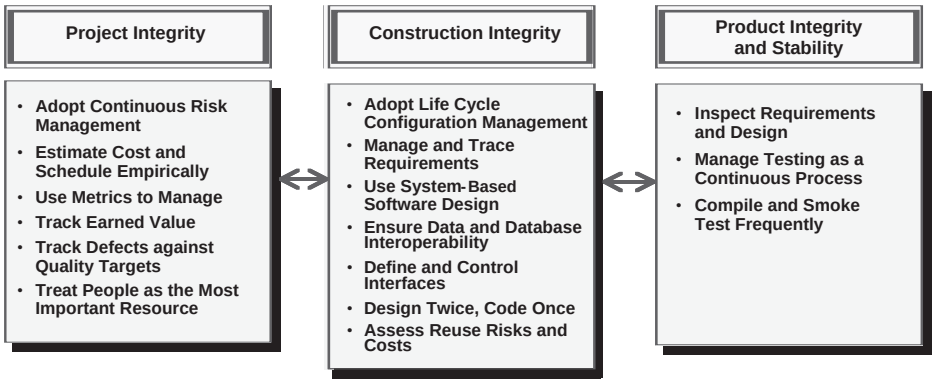
Second, each of the seven factors listed above relates to cultural rather than technical issues, which as previously noted, "Cultural problems are harder to solve than technical problems …" [3]. To address these problems adequately, a manager must understand what makes his or her project function effectively. That is, the manager must answer questions such as, "How do all the project stakeholders interact? What motivates them? Why don't they address important issues even though they are essential to project success?" Only after obtaining the answers to these questions can a manager understand how these seven factors affect the project and then effectively minimize them. For an untrained manager, or a manager under pressure, this is a difficult prospect that often provides more reality than they or their executive management are prepared to deal with.

## Critical Practices

As part of our reassessment of our 2003 observations, we identified several practices that can help mitigate the risks and issues discussed above. The practices we identify here are based on industry standards and have been proven as success criteria in all sizes of programs and projects. These recommended practices would provide a starting point for programs to regain the health of their overall program and provide a high-level road map as a starting point.

One evaluation model is the SPMN 16 Point Plan (Figure 4), which focuses on evaluating critical practices that address

Figure 4: *SPMN 16 Point Plan*



| Practice | Source[1] |
|---|---|
| **Acquisition Best Practices** | |
| Risk management is embraced by the acquisition organization as a cultural imperative and supported and sustained by management. | 16 Point Plan [4] |
| Contract types must match program risk irrespective of administrative load or overhead. | OSD Study [5] |
| A metrics-based reporting structure based on PSM and the SPMN metrics process is defined and written into the contract with severe penalties for misreporting. | COTS Acquisition Study [6] |
| Award fee payments are based on the timely identification and correction of issues rather than the accurate reporting of their existence. | COTS Acquisition Study [6] |
| **Acquisition Characteristics and Infrastructure** | |
| Independent estimation organizations use a calibrated model to evaluate and validate cost and schedule baselines based on worst-case scenarios prior to every acquisition review. | 16 Point Plan [4] |
| Acquisition programs have an active user program that involves the customers and users from the start of the program through deployment and shares the real state of the program, risks, and issues that could preclude success. | Governance Practices [7] |
| Acquisition organizations require structured inspections involving their products and require and pay for contractors/developers to inspect and report metrics concerning defects in requirements, architecture code and other product related components. The acquirer should inspect acquisition products released to developers, as the developer should inspect development products released to the acquirer. The acquisition inspections will find defects in acquirer's products such as concept, user requirements, interfaces, etc. Finding and fixing these defects prior to their use by a developer will have a significant effect at lower rework cost late in the program. | 16 Point Plan [4] |
| A defect profile is negotiated as part of the contract and meeting it is a key part of award fee calculation (with appropriate safeguards). | PSM [8] |
| **Attitudes and Culture** | |
| Practices required by the acquisition organization are planned, executed, and managed by the acquisition organization and not relegated to the supplier. | 16 Point Plan [4] |
| Specific requirements to ensure conformance with enterprise data and process models, including content as well as structure are included in the Contract and Statement of Work. | 16 Point Plan [4] |
| Management and acquisition culture is reality-based, rewarding openness and anticipation of problems and heavily penalizing burying or not seeing risks, issues, or problems that impede success. | COTS Acquisition Study [6], 16 Point Plan [4], OSD Study [5], Governance Practices [7] |
| Managers are rewarded or penalized based on how they address risk and reality during acquisition. | COTS Acquisition Study [6], 16 Point Plan [4], OSD Study [5], Governance Practices [7] |
| The acquisition organization pays for, and requires payment to contractor staff, incentives relating to teambuilding, performance, product completion, and tenure on project. | 16 Point Plan [4] |

Table 1: *Best Practices Matrix*

key, high-leverage areas practiced by successful commercial software developers. These practices pertain to management and control the software development aspects of the work so that the government's requirements are met and high-quality software is delivered on schedule, on time, and within cost.

The 16 Point Plan addresses three primary areas of product management: project integrity, construction integrity, and product integrity and stability. Project integrity encompasses those practices that result in identification of basic project constraints, expectations, and metrics as well as practices used to plan and implement a project environment to predictably satisfy those constraints, expectations, and metrics. Construction integrity encompasses those activities that specify the basic product requirements; maintain traceability to these basic requirements; and control the content, change, and use of the many artifacts and deliverable products that are produced to satisfy user and customer requirements and expectations. The third area, product integrity and stability, ensures that defects (which are inserted in products as part of the software process) are identified and removed in a timely fashion, and that testing is complete and effective and results in the right product consistent with agreed-to requirements and actual expectations.

Acquisition best practices are different than those used for product development, and it is not enough to simply implement a practice that development organizations use such as the SPMN 16 Point Plan. The practices described in Table 1 enable the organization to monitor the developer and receive a product rather than directly monitoring the developing organization that is producing a product. Practices such as integrated risk management, which are critical and must be addressed, should be based on metrics, should maintain visibility into contractor processes, and should evaluate requirements from the acquirer's rather than the developer's perspective.

The practices listed in Table 1 can be misused or misapplied in regard to acquisition practices. For example, the type of contract selected has a bearing on the type of practices to be used and on how they must be adapted. We observed in several assessments during 2003 that the contracting organization was overworked and did not have time to construct or administer a cost-plus fixed-fee (CPFF) contract, despite the fact that a CPFF contract was more suitable to the risk. This situation came about because the contracting professionals did not have a stake in the success of the program but only in the successful award and administration of the contract.

Constrained by the terms and conditions of the contract, the development organization is thus forced to perform high-risk activities such as requirements analysis, architecture development, and defect analysis under an inappropriate contract type. These activities are considered to be high risk because they are difficult and expensive to accomplish late in the program, the findings may result in unanticipated rework not considered under the contract type and necessitate corrective actions that are difficult to complete within the current process, and they are subject to schedule constraints. Correcting these problems would have been much easier had the contract type enabled or supported the flexible process definition. Thus, the wrong contract type can lead to shortcuts, tradeoffs, and decisions based on the cost of the contract rather than the quality of the product.

## Summary

The application of proven best practices by acquisition organizations is a powerful risk reducer. Not all managers and stakeholders who acquire software products have the expertise, training, or incentives to deal with the day-to-day realities of a major acquisition program. As Watts Humphrey put it, "Poor project management will defeat good engineering, and is the most frequent cause of project failure [9]." Managers who use proven best practices that are adapted to the quirks, commitments, and realities of their acquisition program have an advantage that will allow them to anticipate and address the real problems they will invariably face. Rather than rely on *silver bullets* to resolve crises, organizations must establish a culture, based on practices that have been used successfully in the past that anticipates acquisition risks rather than reacts to them. "Enterprises that succumb to the silver bullet syndrome tend to never improve at all, and indeed often go backwards [3]."

Improving acquisition processes works to a point. Most programs have processes, even though their execution is often pro forma. The most effective best practices for acquisition take into consideration the organizational culture. Effective acquisition strategies embrace the uncertainty and risk associated with changing established processes. Acquisition organizations must make the often-significant investment necessary to implement and support the practice (which entails planning, tailoring, practice documentation, method and tool selection, training, productivity impacts, artifact conversion, etc.). Managers must also realize that the new practice may not provide the promised improvement in productivity in the short term. The promise is long term.◆

## References

1. Lister, Tim. "Software Management For Adults." Software Technology Conference, Salt Lake City, UT, 1996.
2. Evans, Michael, Alex Abela, and Tom Beltz. "Seven Characteristics of Dysfunctional Software Projects." CROSSTALK Apr. 2002: 16-20 <www.stsc.hill.af.mil/crosstalk/2002/04/evans. html>.
3. Jones, Capers T. Assessment and Control of Software Risks. New Jersey: Prentice Hall, Feb. 1994: 241.
4. Software Program Managers Network. "16 Critical Software Practices for Implementing Performance-Based Management." Vers. 3.0. Arlington, VA: Integrated Computer Engineering, Inc., 2 Aug. 2000 <www.spmn.com/16CSP. html>.
5. Adams, R., S. Eslinger, K. Owens, and M. Rich. "Software Acquisition Best Practices: 2004 Edition." Third Annual Conference on the Acquisition of Software Intensive Systems, Arlington, VA, 26-28 Jan. 2004 <www.sei.cmu.edu/products/events/acquisition/2004-presentations/adams-eslinger/adams-eslinger.pdf>.
6. Adams, R., and S. Eslinger. "Best Practices for the Acquisition of COTS-Based Software Systems (CBSS): Experiences from the Space System Domain." Ground System Architectures Workshop, Manhattan Beach, CA, Mar. 30 - Apr. 1 2004 <http://sunset.usc.edu/gsaw/gsaw2004/s12/adams_eslinger.pdf>.
7. Dragoon, Alice. "More Governance Best Practices." CIO Aug. 2003 <www.cio.com/archive/081503/factors_sidebar_1.html>.
8. Buys, Ruth T. "DoD Software Core Measures." Fifth Annual PSM Users' Group Conference, Aspen, CO, 23-27 July 2001 <www.psmsc.com/UG2001/Presentations/08OSDSoftwareManagementMetrics.pdf>.
9. Humphrey, Watts S. "Three Dimensions of Process Improvement Part I: Process Maturity." CROSSTALK Feb. 1998: 14-17 <www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/02/processimp.asp>.

## Notes

1. The practices have been modified from the original to reflect the results of the study.

## About the Authors

**Mike Evans** is a senior vice president at American Systems Corporation Inc. Prior to this, he was president of Integrated Computer Engineering, Inc. He is experienced in providing direct technical services and support in software engineering methods and processes, software standards, quality assurance, configuration management, and testing. Evans is co-founder and prime contractor for the Software Program Managers Network, the driving force behind the Department of Defense's software acquisition best practices initiative. He is currently co-writing a book with Dan Galorath on software estimation and risk management issues, due to be published in 2005.

**America Systems Corporation Inc.**
**88674 Shoreline Loop**
**Florence, OR 97439**
**E-mail: candca@aol.com**

**Corinne Segura** is a project manager for American Systems Corporation Inc. (ASC). For the past four years, she has provided technical and managerial support in the areas of process improvement, risk management, program assessments, and test and evaluation. Prior to joining ASC, she served in the U.S. Navy for 20 years as a fleet support officer. Segura has a Bachelor of Science in biology from Northern Illinois University, a Master of Science in systems management from University of Southern California and a Master of Science in electrical engineering from the Naval Postgraduate School.

**American Systems Corporation Inc.**
**3033 5th AVE STE 205**
**San Diego, CA 92103**
**Phone: (619) 421-1595**
**Cell: (619) 208-7140**
**E-mail: corinne.segura@2asc.com**

**Frank Doherty** is the acting deputy program manager for the Intelligence, Surveillance, Reconnaissance, and Information Operations Program Office (PMW-180) at Program Executive Office C4I and Space, San Diego, Ca. Doherty served as lead for acquisition streamlining and chief of industrial quality and productivity division for the Office of the Secretary of Defense, Deputy Director for Contract Administration at U.S. Air Force headquarters; and chief, contract pricing and financial services division, headquarters, Air Force Systems Command.

**PMW-180A**
**Program Executive Office**
**C4I and Space**
**4301 Pacific HWY**
**San Diego, CA 92110-3127**
**Phone: 619-524-7348**
**E-mail: francis.doherty@navy.mil**

# Risk Management (Is Not) for Dummies[1]

Lt. Col. Steven R. Glazewski
*Air Force Institute of Technology*

*Software program managers crave a* silver bullet *in the form of a comprehensive checklist of things to watch so the program does not suffer from bad surprises. Highlighted in this article are some prime examples from almost 15 years' experience acquiring software in Department of Defense programs, from identifying broad areas where software risks tend to hide to describing an eight-step risk management process. While there are no silver bullets to be found, there are a few* golden nuggets *if you make the focused effort to look!*

What is risk management? We have all heard the saying, "Give a man a fish, and you feed him for a day. Teach a man to fish, and you feed him for a lifetime." Let me revise that from a risk management standpoint: "Put out a manager's fires, and you help him for a day. Teach a manager fire prevention, and you help him for a career." If a manager understands good risk management, he can worry about things other than firefighting.

Unfortunately, most people who look for risk management help are seeking to know the steps to put fires out. After all, being a good firefighter has its rewards! Take a look at your organization's person-of-the-quarter listing for the past few years. Who is on it? Typically listed is the person who put out the worst fire. What about people who avoided the fires in the first place? Therein lie the problems with good risk management: *people who avoid fires* do not get noticed, and the risks they *avoid* do not get documented.

Risks that are well understood and controlled tend not to become full-blown problems, and thus are rarely documented in risk databases. To this day, some people mistakenly believe the millions of dollars spent on Year 2000 mitigation were wasted because "nothing bad happened." This is the irony: If people die or property is destroyed, then preventative measures are deemed inadequate; if nobody is hurt and nothing is destroyed, then preventative measures are deemed valueless!

We can do a lot of damage in the name of *process* and *standardization*. Some things lend themselves well to both such as building a car on an assembly line. Some things do not such as creative, knowledge-based work like design and management. Yet we sometimes delude ourselves by creating *templates* for something like a risk management plan. Look carefully at such templates: 80 percent of the outline tends to be *boilerplate* or context setting. The meat is contained in sections that comprise only 20 percent of the table of contents entries. What does the template tell you about those *meaty* sections? Almost nothing! The real meat of a risk management plan – assembling a qualified team, devising ways to discover risks, devising methods of quantifying or categorizing the risks, and monitoring the risks – cannot be completed by simply following a checklist.

In contrast, template instructions for the non-meaty sections tend to be far more explicit (e.g., "state your funding authorization by appropriation for each fiscal year"). Usually, this information is readily available and easily culled from program management plans, status reports, organizational charts, etc. We delude ourselves into thinking that a plan is 80 percent complete when in fact we are just getting started.

> *"Risks that are well understood and controlled tend not to become full-blown problems, and thus are rarely documented in risk databases."*

There is a subtle yet critical message implied in the above: Nobody can give you a simple risk checklist. The reality is, when people want to learn/know *how to do risk management*, they are looking for Dick-and-Jane instructions for the *meaty* 20 percent. That is, they are specifically looking for detailed steps on those things that *cannot* be determined in advance by someone who is not intimately familiar with the project and its domain and environment. Simply put, they are looking for *steps*, words like "go to the financial department and get last month's numbers and look for expenditures that lag the fiscal year spending plan." They do not want *tasks* like "monitor the expenditures to verify claimed accomplishments." The message I get is, "Do not tell me what to think about or investigate, tell me *exactly* who to see, *exactly* what to ask, *exactly* what to record, and *exactly* what to do about it. Don't make this hard – just tell me exactly what to do."

There *can* be value added from a template. But this is far more likely when the template is based on a process or procedure that is absolutely relevant to the program. For example, if you are managing an avionics modernization project, your risk plan template should come from another avionics modernization project. Not only that, but also the template should have been assessed and revised by the last project. This feedback loop is critical! If there was no feedback, then you have no idea if the template's prior users benefited from it or not. In the worst case, the very template you propose to use may have *hindered* their ability to discover, quantify, categorize, prioritize, and manage project risks – and you do not know that! Ideally, the prior users reviewed and updated their risk management plan throughout their project, and all of their lessons learned were captured – you should do this, too.

Speaking of lessons learned, I am often asked for databases of risks, or more simply, where an interested party should look for risks experienced in past programs. The answer always disappoints the inquirer for two reasons. First, the historical data that exists is typically a list of *problems*, not *risks*. Risks are undesirable events that could happen: The concern over possible glitches associated with Year 2000 is a great example. Problems are risks that came to fruition. Problems are well documented in post-mortem analyses. But

good risk management – risk that did not turn into problems – is forgotten.

Second, risks – and even problems – experienced by past programs are *tuned* for the environment that existed for that program and the unique circumstances of that program. What may have been a high-priority risk for a past program may not be worth your investment of resources to monitor or track. Most people who request lessons learned do not really want a database anyway. They want the 15 or 20 items from the database that are most likely to happen to them. And they do not want to read hundreds of items to find those 15 to 20 nuggets. They are really asking me for a five-minute answer to a two-week question.

That is not to say that there is no value added in researching history. My experiences show that there is a fertile ground for finding risks – we know this because problems have consistently arisen from these areas. I have learned to focus some risk identification energy on three areas (if they are present in a project): test and integration hardware, interfaces, and reused code.

- Test and integration hardware tends to be a capacity-constraining resource. If you have a system or software integration lab (SIL), you have a potential resource conflict. Many efforts in the program seem to demand SIL time simultaneously, and usually the software developers do not have top priority. I worked on a program where the same test hardware was used to validate test software *and* to test hardware that was about to be sold to the government. Needless to say, the chance to generate revenue trumped the software developer's needs until we were able to prove that the impending delays to the project would negatively affect the contractor's bottom line by more than a little delay in cash flow. While working on a different program, I discovered that the developer's detailed schedules required over 30 hours *per day* in the SIL to meet the schedule. Scheduling tools are great, but they fail when you disable or ignore the resource conflict warnings.
- Interfaces are historically a source of error, and therefore risk. A recent big example was the Mars Climate Orbiter that crashed into the planet in 1999 because one group coded as if the measurement were in feet while the other coded as if it were in meters. Most bugs in a program are problems found while integrating modules or communicating between objects. On a grander scale in systems of systems, the biggest risks are where the independently built systems must interface. System test engineers always praise a good interface control document (ICD) more than the project managers bemoan the ICD's cost. We have a proverb that "good fences make good neighbors" and the same is true in software: If everyone knows the boundary conditions and interfaces, things go much smoother. The hard part is resisting the temptation to cut or minimize the typically large expense of creating good ICDs. ICDs are used for *inter*-system interfaces, but there are analogous – and equally valuable – design products that should describe the *intra*-system interfaces in detail.
- Reused code, which includes commercial off-the-shelf code, is often *sold* to the program as a means of drastically reducing development and test costs. Code reuse can certainly reduce costs, but only within the very narrow circumstances where you make absolutely no changes to the code, and you use it for *exactly* and *only* the purpose for which it was designed. Many potentially dangerous commercial products like pesticides now carry a standard warning such as "Use of this product in a manner other than described below is a violation of federal law." Yes, the spray is flammable – no, you should not use it to light your barbeque grill. A similar warning should accompany all attempts to reuse code, albeit only a warning that it violates sound reuse strategy, and maybe the laws of good sense. It is not a bad idea to reuse code, but you have to accept the limitations when you do. If your plans call for reusing code and you are assuming substantial time and cost savings or test simplification, you had better not tinker with the reused code (or code products) in any way, or you violate your plan/assumption and incur risk.

Of course, the risk manager must look beyond these three areas, and must apply knowledge of the project's details to determine whether any of those three areas are applicable and worthy of investing resources.

Risk management is much like being the manager of a mutual fund or a stock analyst on Wall Street. Risk managers are asked to peer into the future – to make predictions with better-than-average accuracy – to not only *be* right, but to know what to do when they *are* right. Risk management goes beyond predicting risk; it also demands planning to handle the risk once it materializes. (As a side note, think of how well paid mutual fund managers and Wall Street stock analysts are, especially the successful ones!)

How do fund managers and analysts become successful? They dig into the details of a company. They may not have complete data because the company may not release any more than the minimum required by law. Yet the manager can assemble current information about this particular company, as well as information from its recent and not-so-recent past. Information can be gathered about similar companies over time, and about the segment of the economy that affects this company. This information can then be used to make an educated guess at future earnings, profits, and trends. In other words, they develop detailed knowledge about the specific company, and compare it with a solid general knowledge about the industry and the economy. This helps them more accurately foresee profitability, which can then be used to make sound investment decisions.

This is the essence of risk management! The risk manager combines detailed knowledge of the project with general knowledge of the technical domain and the acquisition environment to foresee potential undesirable events, and to plan and take actions accordingly.

Asking a complete novice to do risk management is, well, risky. Risk management involves thoughtful, determined, and creative work to implement the following eight-step process.

## Step 1: Get Time to Do Risk Management

If you are spending 95 percent of your time doing day-to-day operations, you do not have enough time to sit and think (or plan or just be creative). You need slack time – that is, time away from operations – to plan and think. For a great discussion on why, read Tom DeMarco's book *Slack* [1]. It even contains a few chapters on risk management. Sometimes, this seemingly simple step can be the hardest part. Next comes the creative part.

## Step 2: Plan Your Risk Management Program

What method will you use to discover/elicit risks? Who will help? (Hint: you need those people who are intimately familiar with the project, the domain, and the environment.) What are the desired outputs of your risk analysis? How will you categorize or quantify risk? What information must be recorded for each

risk? Who will use the data and how? Now comes more creativity (problem solving) and some tedium.

## Step 3: Identify Risks

Gather the team and identify potential risks. Remember that the team should consist of people with lots of project and domain experience. These people tend to be senior members and are very busy, so these identification sessions should be short and controlled. Excellent administrative support is absolutely necessary! So is follow-up and coordination of results. For each risk identified, the team should describe what data they need to assess the risk. Much of that data will probably *not* be available at this meeting, which is okay. This first session is identification only.

## Step 4: Assess Risks

The risk team does risk assessment. It involves a facilitator doing lots of research and legwork before another meeting with the experts. Once the data is available and pre-distributed, the team can reconvene to assess probabilities and impacts, determine indicators that a risk may be *coming true*, and prioritize the risks according to the documented procedure. The indicators are used to select metrics so the decision-maker can be proactive when choosing whether to implement handling strategies.

## Step 5: Plan to Handle Risks

With the decision-maker and the team, decide how each risk will be handled. Determine what, if any, mitigation efforts are prudent; what alternative approaches or procedures are available; and/or how to share the risk. It is a good idea to identify thresholds (or trigger points) associated with the metrics selected in Step 4 so it is easier to initiate action.

## Step 6: Monitor Risks

Conduct operations and periodically check to see if any of the risks show signs of turning into problems, or if any of the risks change because of the dynamics of project and environment. This period could be daily or weekly or something different, depending on how dynamic the project and environment are.

## Step 7: Account for Changes in the Environment and Project

Periodically go back to Step 3. This period could be weekly or monthly or something different, depending on how dynamic the project and environment are.

## Step 8: Improve Your Risk Management Process

Periodically go back to Step 2. This period could be quarterly or annually or something different, depending on how successful your program is at giving sufficient notice of things that may go wrong. This is the part that everyone hates, but it is the critical feedback loop that improves the process – for you and for the next project that uses your project as a template.

## General Ideas

Here are some general ideas on risks. They must be general because I do not (and cannot) know the details of every reader's situation.

- If you cannot assign a probability, assess an impact, or draft a unique action plan, then the risk you have identified is too generic, or not a risk at all. For example, stating that the risk is "our budget will get cut" is meaningless because you cannot say what the impact is or what you would do about it. A better risk would be "next year's budget will be cut by 5 percent, which means we cannot fully fund long-lead spares." Document why you chose the numbers you did. Why 5 percent and not 8 percent or 2 percent? Why impact spares and not tech orders?
- If a risk is a near-certainty, then it is not a risk, it is something that the project's execution plan should already address. Does it?
- Risks should be prioritized according to an agreed-upon scheme. The risk team may track 100 risks. Project managers may only have time to track the top 10. Of those, the senior acquisition officials probably have time and attention for only the top two or three. Know how these lists will be derived. Are they based on probability of occurrence? Are they based on severity of impact if they do occur? Are they based on some combination of the two?
- A top 10 list should have exactly 10 items. Having 15 different *No. 1 priority* items may look good when spreading the wealth for performance review bullets, but it does nothing for helping senior people prioritize their time and the *favors* they would like to call in.
- Good risk descriptions include indicators, or some method of foreseeing that the risk may actually be coming true. The better these indicators are, the better you can prepare the contingency plans.

Finally, there are many approaches and processes to manage risk. An Internet search will turn up dozens. But remember the rule of domain applicability: If the risk management process was built by those making and assembling automobiles, it may not be well suited for a different environment such as software development. Risk management, when done correctly, consumes the time of the most experienced, most project-knowledgeable people who also happen to be the busiest and highest-paid. However, the cost and effort to prevent a fire is almost always far less than the cost and effort to rebuild after the fire is out.◆

## Reference

1. Demarco, Tom. Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency. Broadway, 9 Apr. 2002.

## Note

1. The views expressed in this article are those of the author and do not necessarily reflect the official policy or position of the Air Force, Department of Defense, or the U.S. government agency.

## About the Author

**Lt. Col. Steven R. Glazewski** is an instructor at the Air Force Institute of Technology. He teaches Professional Continuing Education courses in software project planning and execution, and software system maintenance as part of the Software Professional Development Program. Glazewski has more than 18 years in weapon system acquisition, including assignments acquiring software for the Advanced Cruise Missile, Embedded GPS/INS navigation unit, and C-5 Avionics Modernization Program, as well as experience maintaining an accredited computer model/simulation. He is an Institute of Electrical and Electronics Engineers Computer Society Certified Software Development Professional.

**Air Force Institute of Technology**
**3100 Research BLVD**
**Pod 3**
**Kettering, OH 45420-4022**
**Phone: (937) 255-7777 ext. 3274**
**DSN: 785-7777 ext. 3274**
**E-mail: steven.glazewski@afit.edu**

# How Do You Make a Peanut Butter and Jelly Sandwich?

The question, "How do you make a peanut butter and jelly (PB&J) sandwich?" takes process back to its basic form: It is a popular question that has been asked by teachers and professors for years in an attempt to teach students how to document step-by-step instructions.

So, what is the process for making a PB&J sandwich?

- Take two slices of bread out of the bag.
- Place peanut butter on one slice of bread.
- Place jelly on the other slice of bread.
- Place the slice of bread with peanut butter on top of the slice of bread with jelly – condiment sides together.
- Eat the sandwich.

Details can be added, for example, instructions to take the twist tie off the bread bag, or indicate how much jelly to use, or how to spread, etc. Then there are the *exceptions*, *changes*, or *tailoring* of the process. Some people like their bread with the crust cut off. But when do you cut if off? Do you cut it off prior to or after the condiments have been added? Do you toast the bread? Then there is the ever-popular question, "How do you slice the sandwich prior to eating it – in halves or triangles?"

Buzzwords have been swirling around this thing called process for some time, including total quality management, continuous improvement, process improvement, International Organization for Standardization (ISO) 9000, Capability Maturity Model® (CMM®), Six-Sigma, and others. All have the same purpose – to make a higher quality product or service faster, better and cheaper. Being able to respond and adapt quickly to the needs and requests of those in the field is a necessity in our industry. The warfighter is our number one priority!

In the last decade, software developers have been asked to document their processes in a number of various ways, one of which is to become CMM Level 3. What exactly does that mean? Basically it certifies that the way the developer does business, whether it is production of systems or development of software, is repeatable, defined, of high quality, and measurable.

Section 804 of the Bob Stump National Defense Authorization Act for Fiscal Year 2003 mandates that government acquisition organizations begin process improvement efforts in-house. Section 804 requires the establishment of software *acquisition process improvement* (API) programs by those defense agencies that manage major defense acquisition programs with a substantial software component. The API requirements include the following:

- Documented processes.
- Appropriate metrics to verify performance and acquisition process improvement.
- Ensuring appropriate training or experience.
- Ensuring adherence to processes and requirements.

By starting a process improvement effort in-house, government acquisition organizations complement the efforts being accomplished by their Level 3 developers. Most government acquisition organizations do not produce systems or develop software – they manage, monitor, and acquire these services from others.

To support organizations that *acquire* products/services, the Software Engineering Institute (SEI) created the Software Acquisition Capability Maturity Model (SA-CMM) to complement the CMM for Software (SW-CMM) and the Systems Engineering CMM (SE-CMM). The SA-CMM includes both systems and software and is a framework for improving acquisition processes, describing the *buyers* role. The model is used by senior management to set goals and to assess an organization's maturity. Its use is appropriate throughout the entire product life cycle.

With the CMM Integration℠ replacing the SW-CMM and SE-CMM, the office of the secretary of defense has requested that the SEI assist in developing a CMMI Acquisition Module. Currently, this document is in draft form with pilots and a final version is due this year. The module does not have levels; instead, it concentrates on continuous process improvement rather than the need to acquire a level. The proposed CMMI Acquisition Module is based on the CMMI model, incorporating best practices from the SA-CMM, the Federal Aviation Administration's i-CMM, Section 804, and other sources. It is streamlined (only 32 pages), easily implemented through self-assessments, and does not require an extensive infrastructure. The module focuses on effective acquisition activities and practices that are implemented by first-level acquisition projects.

Keep in mind, when you ask your child to write up the PB&J sandwich steps, it is his/her job to give you grief about it, and your job to embarrass him/her with a picture of him/her with peanut butter in the hair and jelly up the nose.

Remember, when your processes are documented and after you make your sandwich you get to enjoy it, but BYOPB (bring your own peanut butter).

— Mamie Danley Morgan, MSOD
Senior Systems Engineer
*L-3 Communications, GSI*
mamie.morgan@L-3com.com

## Can You BACKTALK?

Here is your chance to make your point, even if it is a bit tongue-in-cheek, without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CROSSTALK, we also accept articles for the BACKTALK column. BACKTALK articles should provide a concise, clever, humorous, and insightful perspective on the software engineering profession or industry or a portion of it. Your BACKTALK article should be entertaining and clever or original in concept, design, or delivery. The length should not exceed 750 words.

For a complete author's packet detailing how to submit your BACKTALK article, visit our Web site at <www.stsc.hill.af.mil>.

**CrossTalk / MASE**
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820



PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737